SDN MAGAZINE

Nummer
149
jan 2024

SDN Magazine is een uitgave van:
SDN

Van en voor Microsoft & .NET professionals

- GraphQL Deel 2
- Semantic Kernel
- Artifical Intelligence and Dail Phones

Colofon

Uitgave
Software Development Network
27ste jaargang
Nr. 149 • januari 2024

Redactiecommissie:
Annejan Barelds, Nadine Wolff, Jan de Vries, John Bruin, Menno Jongerius, Marcel Meijer, Roy Janssen, Roelant Dieben, Thijs den Hollander.

Auteurs:
Annejan Barelds, Arjen de Ruiter, Dhruv Patel, Jan de Vries, Johan Smaris, Nadine Wolff, Sander Hoogendoorn, Stef Heyenrath

Listings:
Zie de website www.sdn.nl voor eventuele source files uit deze uitgave

Contact:
Software Development Network
info@sdn.nl
redactie@sdn.nl

Adverteren:
Informatie over adverteren en de tarieven kunt u opvragen bij
Lieke Stomps
lstomps@reshift.nl

Beste SDN-er,

Het is weer gelukt om een mooi gevuld magazine voor jullie te maken. Dit is ook mijn laatste magazine als eindredacteur en voorzitter van de SDN.

Na ruim 18 jaar doe ik een stap terug en verlaat ik de SDN. Veel dank ben ik verschuldigd aan Joop, Rob en Remi. Zij waren het die mij de mogelijkheden gaven en zoals het hoort heb ik anderen ook mogelijkheden geboden. Ik was spreker, auteur, voorzitter, administrateur, eindredacteur, redacteur, vormgever, social media, community manager, event planner/organisator, gastheer, spreker scout en nog veel meer.

In die 18 jaar heeft de SDN en heb ik verschillende Microsoft community collega's het podium geboden om te starten met schrijven (voor dit magazine) en presenteren op een van onze events. Voor allemaal vormde ik de basis van hun carrière als presentator en bijna allemaal hebben op grote nationale en internationale events (zoals Techdays, DevDays, Techorama, Futuretech, NDC's, grote events in Polen, Engeland, Duitsland etc.). Ook is een groot aantal auteur van een of meerdere belanghebbende boeken geworden. Of zijn zelfs invloedrijke Youtubers geworden. De SDN bood het platform voor startende sprekers of beginnende auteurs. Daar ben ik heel erg trots op!

Martin, Roelant, Kelly en natuurlijk Roy nemen het roer verder over en ik vertrouw erop dat er nog veel mooie events en magazines gaan komen! Op naar de volgende 35 jaar als SDN.

Onze auteurs hebben er weer juweeltjes voor jullie gemaakt en deze keer waren dat: Nadine, Sander, Johan, Jan, Stef, Dhruv en Annejan. Met een column over managers en column over Artificial intelligence, deel 2 van GraphQL, Semantic Kernel, ChatGpt, SQL via natuurlijke taal, Azure API management en een interview over het Technical Debt spel.

Veel leesplezier!

Groeten,
Marcel

DEVITJOBS.NL

PHP Developer

Salarisindicatie:
40.000 - 50.000 EUR

Locatie:
Breukelen

Technologieën:
UX UI Design, API, CRM, PHP

orangevalley

ASP.NET back-end framework developer

Salarisindicatie:
45.000 - 70.000 EUR

Locatie:
Amsterdam

Technologieën:
.NET Framework, ASP.NET, C#, SQL, .NET

ChipSoft

.NET Software Developer - Smart Logistics

Salarisindicatie:
35.000 - 60.000 EUR

Locatie:
Rotterdam

Technologieën:
.NET Framework, ASP.NET, Architect, Blazor, C#, DevOps, ICT, MVC, Product Owner, REST, SQL, WebServices, .NET

CGI

Senior .NET Ontwikkelaar

Salarisindicatie:
55.000 - 80.000 EUR

Locatie:
Barendrecht

Technologieën:
.NET Framework, C#, Azure, .NET

ICT GROUP

Find all these vacancies and more

at Devitjobs.nl

In dit nummer

11 Het zal je niet zijn ontgaan dat Open AI en hun bekende Large Language Model implementaties (zoals ChatGPT) veel aandacht hebben gekregen. Vele lezers zullen hier dan ook gebruik van hebben gemaakt, wellicht ook voor het maken van code. Een andere bekende LLM-implementatie is natuurlijk GitHub Copilot, welke als doel heeft om “natural language” vragen naar software implementaties (code) te vertalen.

23 Now with the availability of large language model capable of solving medium-complex problems, we can leverage its capabilities to generate SQL queries for our natural language asks. In this article, we will see various methods to accomplish this goal. There are some prerequisites you need to know, before starting to develop software using a LLM. Some of these basics are described below.

27 Sinds een aantal jaren is Azure API Management haast onvermijdelijk voor iedereen die API’s ontwikkelt en deployed in Azure. Het is dan ook een fantastische one-stop-shop voor allerlei aspecten die samenhangen met het beheer, de publicatie en de consumptie van API’s, die je écht liever niet in code regelt. Azure API Management functioneert als Gateway, warbinnen je bijvoorbeeld caching, throttling of authenticatie kunt regelen.

GraphQL voor .NET Developers **deel 2**

In de vorige editie van het SDN-magazine heb ik een artikel gepubliceerd over het maken van een GraphQL service in .NET met behulp van de NuGet-package Hot-Chocolate. Deze service was alleen nog niet gekoppeld aan een database. In dit artikel wordt deze koppeling besproken. De database zal worden ontsloten met behulp van Entity Framework Core.

Auteur: Johan Smarius.

Wat was de casus ook alweer?

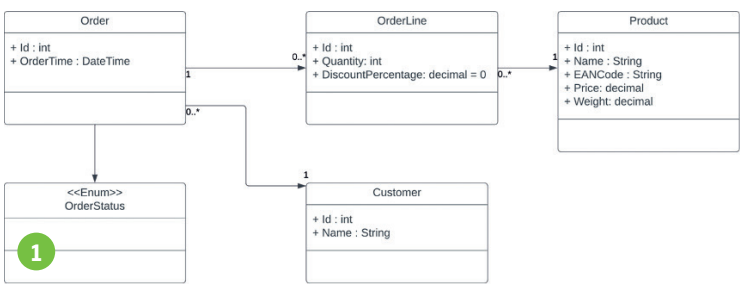
In het vorige artikel is een casus geïntroduceerd van een heel simpel ordermanagement systeem. Dit artikel zal ook gebruik maken van dezelfde casus. Om je geheugen op te frissen of als je het vorige artikel nog niet gelezen hebt, staat het klassendiagram in **Figuur 1**.
Bij het vorige artikel is de Query klasse voor Order gemaakt (**Figuur 2**). In dit artikel wordt verder gebouwd op deze code.

DbContext

Voor gebruik binnen GraphQL worden geen speciale eisen gesteld aan de DbContext die je maakt. De context die voor deze casus gebruikt wordt, is opgenomen in **Figuur 3**.
Binnen deze context wordt de database voorzien van initiële data in de OnModelCreating (**Figuur 4**). Voor deze casus worden 2 orders inclusief afhankelijkheden in de database gezet. Deze context wordt via de normale registratie opgenomen in de service collection van ASP.NET Core (**Figuur 5**). Bij de registratie wordt wel gebruikt gemaakt van een pooled context factory, omdat HotChocolate meerdere parallellen contexten kan gebruiken. De extra logging is alleen aangezet om de query's die uitgevoerd worden inzichtelijk te maken voor dit artikel. Deze logginginstellingen zijn uitdrukkelijk geen best practice voor productiecode.

Integratie van DbContext met HotChocolate

Om de datacontext binnen HotChocolate te kunnen gebruiken, is het nodig om de NuGet-package "HotChocolate.Data.EntityFramework" aan het project toe te voegen (**Figuur 6**). Voor dit artikel wordt



Domeinmodel van het voorbeeld

```
public class OrderQuery
{
    public IEnumerable<Order> GetAllOrders()
    {
        return GenerateTestOrders();
    }

    public Order GetOrderById(int id)
    {
        return GenerateTestOrders().Single(order => order.Id == id);
    }

    private List<Order> GenerateTestOrders(){...}
}
```

2

Query code

```
public class OrderContext : DbContext
{
    public DbSet<Order>? Orders { get; set; }

    public DbSet<OrderLine>? OrderLines { get; set; }

    public DbSet<Customer>? Customers { get; set; }

    public OrderContext(DbContextOptions<OrderContext> options) : base(options)
    {
    }

    protected override void OnModelCreating(ModelBuilder modelBuilder){...}
}
```

3

Code OrderContext

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    base.OnModelCreating(modelBuilder);

    modelBuilder.Entity<Customer>().HasData(
        new Customer() { Id = 1, Name = "Customer 1"},
        new Customer() { Id = 2, Name = "Customer 2"}
    );
    modelBuilder.Entity<Product>().HasData(
        new Product() { Id = 1, Name = "Product 1", EANCode = "1234599980", Price = 100, Weight = 300 },
        new Product() { Id = 2, Name = "Product 2", EANCode = "3703483483 9", Price = 200, Weight = 700}
    );
    modelBuilder.Entity<Order>().HasData(
        new List<Order>()
        {
            new()
            {
                Id = 1, CustomerId = 1, OrderTime = DateTime.Now
            },
            new()
            {
                Id = 2, CustomerId = 2, OrderTime = DateTime.Now
            }
        }
    );
    modelBuilder.Entity<OrderLine>().HasData(
        new OrderLine() {Id = 2, OrderId = 1, ProductId = 1, Quantity = 2, DiscountPercentage = 0},
        new OrderLine() {Id = 1, OrderId = 1, ProductId = 2, Quantity = 5, DiscountPercentage = 5},
        new OrderLine() {Id = 3, OrderId = 2, ProductId = 1, Quantity = 7, DiscountPercentage = 0},
        new OrderLine() {Id = 4, OrderId = 2, ProductId = 2, Quantity = 10, DiscountPercentage = 20.0m}
    );
}
```

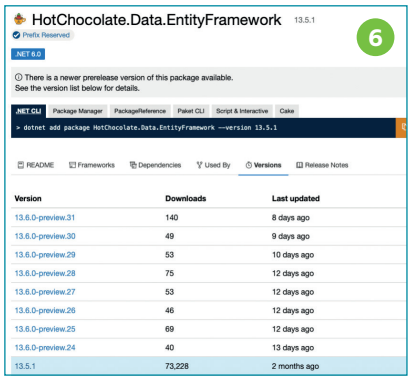
4

Code van OnModelCreating

```
builder.Services.AddPooledDbContextFactory<OrderContext>(
    optionsAction: options => options.UseSqlite(connectionString: "Data Source=Orders.db")
    .EnableSensitiveDataLogging()).AddLogging(Console.WriteLine);
```

5

Registratie OrderContext



NuGet package (bron: https://www.nuget.org/packages/HotChocolate.Data.EntityFramework#versions-body-tab)

uitgegaan van de laatste stabiele versie (13.5.1).
https://www.nuget.org/packages/HotChocolate.Data.EntityFramework#versions-body-tab)
De datacontext moet bekend worden gemaakt aan de GraphQL code door de context te registreren (**Figuur 7**).
Door de code in figuur 5 en 7 wordt de dbcontext nu injecteerbaar gemaakt voor de Query klasse. Binnen ASP.NET Core is het gebruikelijk om dependencies te injecteren in de constructor van een klasse. Deze aanpak levert binnen HotChocolate problemen op, omdat dependencies dan als Singleton behandeld worden en dat is voor een DbContext niet wenselijk. De maker van HotChocolate (Chillicream) adviseert dan ook om de injectie op methodeniveau te doen.
Aan de code voor de query kan een methode toe worden gevoegd om alle orders uit de database op te halen (**Figuur 8**). Voor de goede werking van de server is het noodzakelijk om een IEnumerable of IQueryable terug te geven. De methode krijgt de OrderContext geïnjecteerd en kan via LINQ alle informatie ophalen.
In dit voorbeeld is ervoor gekozen om ook direct alle afhankelijkheden geforceerd te laden. Dit is gedaan om ervoor te zorgen dat deze afhankelijkheden ook beschikbaar zijn om informatie uit op te vragen.

```
builder.Services.AddGraphQLServer()
    .AddQueryType<OrderQuery>()
    .RegisterDbContext<OrderContext>(DbContextKind.Pooled);
```
7

Registratie van DbContext bij GraphQL

```
johansmarius
public IEnumerable<Order> GetOrdersEnumerable(OrderContext orderContext) =>
    orderContext.Orders// DbSet<Order>?
        .Include(navigationPropertyPath: order => order.OrderLines).ThenInclude(orderline => orderline.Product)// IIncludableQueryable<Order,Product>
        .Include(navigationPropertyPath: order => order.Customer);// IIncludableQueryable<Order,Customer>
```
8

Code voor ophalen alle order informatie uit database

```
query {
    ordersEnumerable {
        customer {
            name
        }
        id
        orderLines {
            order {
                orderStatus
            }
            product {
                name
                eanCode
            }
            quantity
        }
    }
}
```
9

Query voor testen nieuwe methode


```
{
  "data": {
    "ordersEnumerable": [
      {
        "customer": {
          "name": "Customer 1"
        },
        "id": 1,
        "orderLines": [
          {
            "order": {
              "orderStatus": "NEW"
            },
            "product": {
              "name": "Product 2",
              "eanCode": "37034034039"
            },
            "quantity": 5
```

10

Deel van het resultaat van de query

Als deze code uitgevoerd wordt, kan bijvoorbeeld de volgende query uit worden gevoerd (**Figuur 9**).
Dit levert dan het volgende resultaat op (**Figuur 10**).
In de logging kan de SQL-query die uitgevoerd wordt, worden teruggevonden (**Figuur 11**). Via GraphQL kan wel aan worden gegeven welke velden getoond moeten worden, maar dit resulteert nog niet in het alleen maar ophalen van deze velden uit de database. Via deze oplossing hebben we dus last van overfetching. Voor dit kleine voorbeeld zal dit niet problematisch zijn, maar bij een productiedatabase met heel veel kolommen per tabel kan dit toch wel tot problemen leiden.

Tegengaan van overfetching

Gelukkig biedt de integratie van HotChocolate met EntityFramework Core hier een passende oplossing voor. De integratie maakt het mogelijk om projecties toe te passen. Deze kunnen worden geactiveerd door boven de methode het attribuut UseProjection op te nemen (**Figuur 12**). Een bijkomend voordeel van het toepassen van dit attribuut

11

Query die uitgevoerd wordt

```
[UseProjection]
johansmarius
public IQueryable<Order> GetOrders(OrderContext orderContext) => orderContext.Orders;
```
12

Gebruik van UseProjection attribuut

```
builder.Services.AddGraphQLServer()
    .AddQueryType<OrderQuery>()
    .RegisterDbContext<OrderContext>(DbContextKind.Pooled)
    .AddProjections();
```

13

Aanpassing aan serviceregistratie

```
Executed DbCommand (13ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
SELECT 1, "c"."Name", "o"."Id", "c"."Id", "t"."c", "t"."OrderStatus", "t"."c0", "t"."Name", "t"."EANCode", "t"."Quantity", "t"."Id", "t"."Id0", "t"."Id1"
FROM "Orders" AS "o"
INNER JOIN "Customers" AS "c" ON "o"."CustomerId" = "c"."Id"
LEFT JOIN (
    SELECT 1 AS "c", "o1"."OrderStatus", 1 AS "c0", "p"."Name", "p"."EANCode", "o0"."Quantity", "o0"."Id", "o1"."Id" AS "Id0", "p"."Id" AS "Id1", "o0"."OrderId"
    FROM "OrderLines" AS "o0"
    INNER JOIN "Orders" AS "o1" ON "o0"."OrderId" = "o1"."Id"
    INNER JOIN "Product" AS "p" ON "o0"."ProductId" = "p"."Id"
) AS "t" ON "o"."Id" = "t"."OrderId"
ORDER BY "o"."Id", "c"."Id", "t"."Id", "t"."Id0"
```

14

Query die naar de database wordt gestuurd na toepassing projecties

in combinatie met het return type van IQueryable is dat het niet meer nodig is om alle dependencies van tevoren te specificeren.
Om dit attribuut te kunnen laten werken, moet nog wel een kleine aanpassing worden gemaakt bij het registreren van de GraphQL service (**Figuur 13**). AddProjections moet hieraan toe worden gevoegd.
Als de code nu wordt gerund en dezelfde GraphQL query wordt uitgevoerd, dan wordt een heel andere SQL-query naar de database gestuurd (**Figuur 14**). Alleen die informatie die nodig is voor de GraphQL query wordt nu ook opgehaald uit de database. Hierbij wordt dus overfetching van kolommen voorkomen.

Toepassen van filtering

Met het toevoegen van projecties kan het overfetchen van kolommen tegen worden gegaan. Helaas geldt dit nu nog niet voor het aantal records dat teruggegeven kan worden door de query. Het is nu nog alles of niets. Het zou dus heel handig zijn als ook filtering toe zou kunnen worden gepast, zodat alleen een specifiek aantal records uit de database opgehaald zou kunnen worden. Gelukkig kan dit ook. Hiervoor kan het attribuut UseFiltering worden gebruikt (**Figuur 15**).
Om dit te kunnen laten werken, moet ook nu de serviceregistratie aan worden gepast (**Figuur 16**). De optie "AddFiltering" moet toe worden gevoegd.
Als deze code nu uit wordt gevoerd, dan kan in de query filtering toe worden gepast (**Figuur 17**).

```
[UseProjection]
[UseFiltering]
johansmarius
public IQueryable<Order> GetOrders(OrderContext orderContext) => orderContext.Orders;
```

15

Toevoegen van extra attribuut UseFiltering

```
builder.Services.AddGraphQLServer()
    .AddQueryType<OrderQuery>()
    .RegisterDbContext<OrderContext>(DbContextKind.Pooled)
    .AddProjections()
    .AddFiltering();
```

16

Toevoegen van filter optie aan serviceregistratie

```
query {
    orders(where: { id: {eq: 1}}) {
        customer {
            name
        }
        id
        orderLines {
            order {
                orderStatus
            }
            product {
                name
                eanCode
            }
            quantity
        }
    }
}
```

17

GraphQL query met filter optie

```
{
  "data": {
    "orders": [
      {
        "customer": {
          "name": "Customer 1"
        },
        "id": 1,
        "orderLines": [
          {
            "order": {
              "orderStatus": "NEW"
            },
            "product": {
              "name": "Product 2",
              "eanCode": "37034034039"
            },
            "quantity": 5
          },
          {
            "order": {
              "orderStatus": "NEW"
            },
```

18

Resultaat van het uitvoeren van de query met filtering

```
Executed DbCommand (0ms) [Parameters=[@__p_0='1'], CommandType='Text', CommandTimeout='30']
SELECT 1, "c"."Name", "o"."Id", "c"."Id", "t"."c", "t"."OrderStatus", "t"."c0", "t"."Name", "t"."EANCode", "t"."Quantity", "t"."Id", "t"."Id0", "t"."Id1"
FROM "Orders" AS "o"
INNER JOIN "Customers" AS "c" ON "o"."CustomerId" = "c"."Id"
LEFT JOIN (
    SELECT 1 AS "c", "o1"."OrderStatus", 1 AS "c0", "p"."Name", "p"."EANCode", "o0"."Quantity", "o0"."Id", "o1"."Id" AS "Id0", "p"."Id" AS "Id1", "o0"."OrderId"
    FROM "OrderLines" AS "o0"
    INNER JOIN "Orders" AS "o1" ON "o0"."OrderId" = "o1"."Id"
    INNER JOIN "Product" AS "p" ON "o0"."ProductId" = "p"."Id"
) AS "t" ON "o"."Id" = "t"."OrderId"
WHERE "o"."Id" = @__p_0
ORDER BY "o"."Id", "c"."Id", "t"."Id", "t"."Id0"
```

19

SQL logging na het toepassen van filtering

Na het uitvoeren van deze query wordt ook maar 1 order teruggegeven (**Figuur 18**).
En niet alleen wordt alleen maar 1 record getoond, maar de logging toont ook aan dat er maar 1 record ook uit de database op wordt gehaald (**Figuur 19**).
De filtering wordt gestart door het keyword "where". Achter dit keyword

wordt aan de hand van een json syntax de filterconditie opgegeven. In het voorbeeld is gebruik gemaakt van een simpele conditie op basis van id, maar de mogelijkheden om te filteren zijn veel uitgebreider. Zo is in **Figuur 20** te zien dat bijvoorbeeld ook gefilterd kan worden op klantnaam. Voor het specificeren van de conditie kan dus het navigatiepad binnen het GraphQL-schema worden gebruikt.

```
query {
    orders(where: { customer: {name: { eq: "Customer 2"}}}) {
        customer {
            name
        }
        id
        orderLines {
            order {
                orderStatus
            }
            product {
                name
                eanCode
            }
            quantity
        }
    }
}
```

20

Filtering op basis van klantnaam

De filtering kan eventueel ook genest worden toegepast. Binnen de code voor de Order is het attribuut "UseFiltering" toegepast voor de OrderLines collectie (**Figuur 21**).

Deze aanpassing maakt het mogelijk om bijvoorbeeld de volgende GraphQL-query uit te voeren (**Figuur 22**). Hierbij worden alleen die orderlines opgehaald waarvan het orderaantal groter is dan 4. Rekening houdend met de seed die toegepast is, moet dit resulteren in maar 1 orderline die opgehaald wordt.

```
public class Order
{
    5 usages
    public int Id { get; set; }

    [UseFiltering]
    5 usages
    public ICollection<Orderline> OrderLines { get; set; } = new List<Orderline>();

    3 usages
    public Customer Customer { get; set; }
    2 usages
    public int CustomerId { get; set; }

    4 usages
    public DateTime OrderTime { get; set; }

    2 usages
    public OrderStatus OrderStatus { get; set; }
}
```

21

Code-aanpassing aan OrderLines

```
query {
    orders(where: { id: { eq: 1} }) {
        customer {
            name
        }
        id
        orderLines(where: { quantity: { gt: 4}}) {
            order {
                orderStatus
            }
            product {
                name
                eanCode
            }
            quantity
        }
    }
}
```

22

Geneste query

```
Executed DbCommand (0ms) [Parameters=[@__p_1='4', @__p_0='1'], CommandType='Text', CommandTimeout='30']
SELECT 1, "c"."Name", "o"."Id", "c"."Id", "t"."c", "t"."OrderStatus", "t"."c0", "t"."Name", "t"."EANCode", "t"."Quantity", "t"."Id", "t"."Id0", "t"."Id1"
FROM "Orders" AS "o"
INNER JOIN "Customers" AS "c" ON "o"."CustomerId" = "c"."Id"
LEFT JOIN (
    SELECT 1 AS "c", "o1"."OrderStatus", 1 AS "c0", "p"."Name", "p"."EANCode", "o0"."Quantity", "o0"."Id", "o1"."Id" AS "Id0", "p"."Id" AS "Id1", "o0"."OrderId"
    FROM "OrderLines" AS "o0"
    INNER JOIN "Orders" AS "o1" ON "o0"."OrderId" = "o1"."Id"
    INNER JOIN "Product" AS "p" ON "o0"."ProductId" = "p"."Id"
    WHERE "o0"."Quantity" > @__p_1
) AS "t" ON "o"."Id" = "t"."OrderId"
WHERE "o"."Id" = @__p_0
ORDER BY "o"."Id", "c"."Id", "t"."Id", "t"."Id0"
```

23

SQL Logging voor genest query

```
[UseProjection]
[UseFiltering]
[UseSorting]
johansmarius
public IQueryable<Order> GetOrders(OrderContext orderContext) => orderContext.Orders;
```

24

Toevoegen UseSorting attribuut

Als deze GraphQL-query uit wordt gevoerd, dan blijkt uit de logging van EF Core ook dat inderdaad alleen maar die orderlines op worden gehaald waarvan het aantal groter is dan een bepaalde parameterwaarde (**Figuur 23**).

Toepassing van filtering maakt het dus mogelijk om binnen GraphQL query's te schrijven om zo heel specifiek te maken welke data opgehaald moet worden. Binnen HotChocolate kunnen nog veel meer filters gebruikt worden. Meer informatie over alle filtermogelijkheden staat beschreven op: https://chillicream.com/docs/hotchocolate/v13/fetching-data/filtering.

Sortering

Binnen applicaties is het gebruikelijk om sortering toe te passen. Natuurlijk kan dit ook clientside worden gedaan, maar het is zeker bij grote datasets veel handiger om dit ook serverside te kunnen doen. GraphQL ondersteunt dit ook. Net als bij de andere opties kan dit aangegeven worden door een attribuut toe te passen (**Figuur 24**) en

de optie toe te voegen aan de service-registratie (**Figuur 25**).
Als deze code nu uitgevoerd wordt, dan kan de sortering aangegeven worden door het keyword order te gebruiken (**Figuur 26**). Per veld kan aan worden gegeven of het op- of aflopend gesorteerd moet worden. De eerste order die nu getoond wordt, is nu de tweede order, omdat de klantnaam daarvan "Customer 2" is (**Figuur 27**).

```
builder.Services.AddGraphQLServer()
    .AddQueryType<OrderQuery>()
    .RegisterDbContext<OrderContext>(DbContextKind.Pooled)
    .AddProjections()
    .AddFiltering()
    .AddSorting();
```
25

Toevoegen van sortering aan serviceregistratie

```
query QueryById {
    orders(order: {customer: { name: DESC} }) {
        customer {
            name
        }
        id
        orderLines {
            order {
                orderStatus
            }
            product {
                name
                eanCode
            }
            quantity
        }
    }
}
```
26

GraphQL query met sortering

```
{
  "data": {
    "orders": [
      {
        "customer": {
          "name": "Customer 2"
        },
        "id": 2,
        "orderLines": [
          {
            "order": {
              "orderStatus": "NEW"
            },
            "product": {
              "name": "Product 1",
              "eanCode": "123439900"
```
27

Resultaat van de GraphQL query met sortering

```
[UseOffsetPaging]
[UseProjection]
[UseFiltering]
[UseSorting]
johansmarius *
public IQueryable<Order> GetOrders(OrderContext orderContext) => orderContext.Orders;
```
28

Toevoegen van paginering

```
builder.Services.AddGraphQLServer()
    .AddQueryType<OrderQuery>()
    .RegisterDbContext<OrderContext>(DbContextKind.Pooled)
    .AddProjections()
    .AddFiltering()
    .AddSorting()
    .SetPagingOptions(new PagingOptions() { DefaultPageSize = 1, MaxPageSize = 1 });
```
29

Toevoegen van paginering aan de serviceregistratie

Paginering

Een laatste optie die vaak nodig is bij het maken van GraphQL services, maar eigenlijk voor elke back-end service in het algemeen, is de mogelijkheid om paginering toe te passen. HotChocolate biedt hier ook ondersteuning voor. De optie kan per methode weer via een attribuut aan worden gezet (**Figuur 28**).
Bij het toevoegen van de optie aan de lijst van attributen moet nu wel goed op de volgorde worden gelet. De pagineringsoptie moet als eerste worden opgegeven. Bij dit voorbeeld wordt gebruik gemaakt van offset paginering. Deze pagineringsaanpak maakt gebruik van skip en take opties, zoals ook door Entity Framework Core worden ondersteund.
De registratie van de paginering verloopt net even wat anders dan de andere opties (**Figuur 29**). De opties voor de pagesize kunnen hierbij ingesteld worden.
Als deze code nu uit wordt gevoerd, dan ziet de GraphQL query er een beetje anders uit (**Figuur 30**).
Het resultaat bevat nu ook informatie of er nog een vorige en/of volgende pagina op de server beschikbaar is (**Figuur 31**). Omdat het eerste record overgeslagen is, is er nog een vorige pagina. De database bevat maar 2 orders, dus er is geen sprake meer van een volgende pagina.

Conclusie

Met behulp van HotChocolate en EntityFramework Core kan op een eenvoudige manier een GraphQL server in .NET worden gemaakt die gekoppeld is aan een database. Dit is natuurlijk niet de enige mogelijkheid, maar ik vind het wel een van de simpelste. In de volgende editie van het SDN Magazine zal de mogelijkheid besproken worden om ook mutaties door te voeren door middel van GraphQL.
Op https://github.com/johansmarius staat de code die gebruikt is in dit artikel. De screenshots in dit artikel zijn afkomstig uit deze code. ●

```
query {
    orders(take: 1, skip: 1) {
        pageInfo {
            hasNextPage
            hasPreviousPage
        }
        items{
            customer {
                name
            }
            id
            orderLines {
                order {
                    orderStatus
                }
                product {
                    name
                    eanCode
                }
                quantity
            }
        }
    }
}
```
30

GraphQL query met pagineringsinformatie

```
{
  "data": {
    "orders": {
      "pageInfo": {
        "hasNextPage": false,
        "hasPreviousPage": true
      },
      "items": [
        {
          "customer": {
            "name": "Customer 2"
          },
          "id": 2,
          "orderLines": [
            {
              "order": {
                "orderStatus": "NEW"
              },
              "product": {
                "name": "Product 1",
                "eanCode": "123439900"
              },
              "quantity": 7
            },
            {
```
31

Resultaat van GraphQL query met paginering

De volgende stap in development met Semantic Kernel

Het zal je niet zijn ontgaan dat Open AI en hun bekende Large Language Model implementaties (zoals Chat-GPT) veel aandacht hebben gekregen. Vele lezers zullen hier dan ook gebruik van hebben gemaakt, wellicht ook voor het maken van code. Een andere bekende LLM-implementatie is natuurlijk GitHub Copilot, welke als doel heeft om "natural language" vragen naar software implementaties (code) te vertalen.

Auteur: Jan de Vries

Dit is allemaal indrukwekkend als gebruiker, maar het voegt nog niet direct waarde toe voor de oplossingen die we voor onze bedrijven maken. Als ontwikkelaar kunnen we de verschillende end-points (completion, chat, embedding) aanroepen van de Open AI instanties, maar om daar nuttige output uit te krijgen dient er goede input (prompt) te worden gemaakt. Dit integreren in een bestaande oplossing kan voor uitdagingen zorgen, plus dat het maken van dergelijke input wel een vak apart blijkt te zijn.
Voor een goede implementatie van een LLM in jouw applicatie is het noodzakelijk om anders te gaan denken over softwareontwikkeling en dit op een specifieke manier op te zetten.

Wat gaat Semantic Kernel voor mij doen?

Het is relatief eenvoudig om te werken met Azure Open AI, deze heeft namelijk verschillende endpoints beschikbaar om acties uit te voeren. Een voorbeeld, uit de MS Learn documentatie, hoe het completion endpoint van GPT-3.5 Turbo aangeroepen kan worden met de C# SDK. **Listing 1**
Uiteraard is bovenstaande voorbeeld code, maar geeft al goed weer hoe Azure Open AI geïntegreerd kan worden in bestaande software.
Helaas wordt de chat-ervaring heel veel gebruikt om de kracht van een LLM te laten zien. In veel software-oplossingen is dit niet een ervaring die wordt aangeboden of nuttig wordt geacht.

Ook zullen de meeste oplossingen meer doen dan alleen informatie vergaren uit een LLM. Normaliter zal dat in een of meerdere databronnen zijn opgeslagen, zijn er algoritmes ontwikkeld om waarde toe te voegen aan de data, etc.
Het samenvoegen van de mogelijkheden die een LLM biedt en een bestaande softwareoplossing is iets waar frameworks zoals Semantic Kernel in uitblinken. Een ander bekend framework, dat een vergelijkbaar doel heeft is LangChain. Een quote van de LangChain site:
LangChain is a framework for developing applications powered by language models. We believe that the most powerful and differentiated applications will not only call out to a language model via an api, but will also:

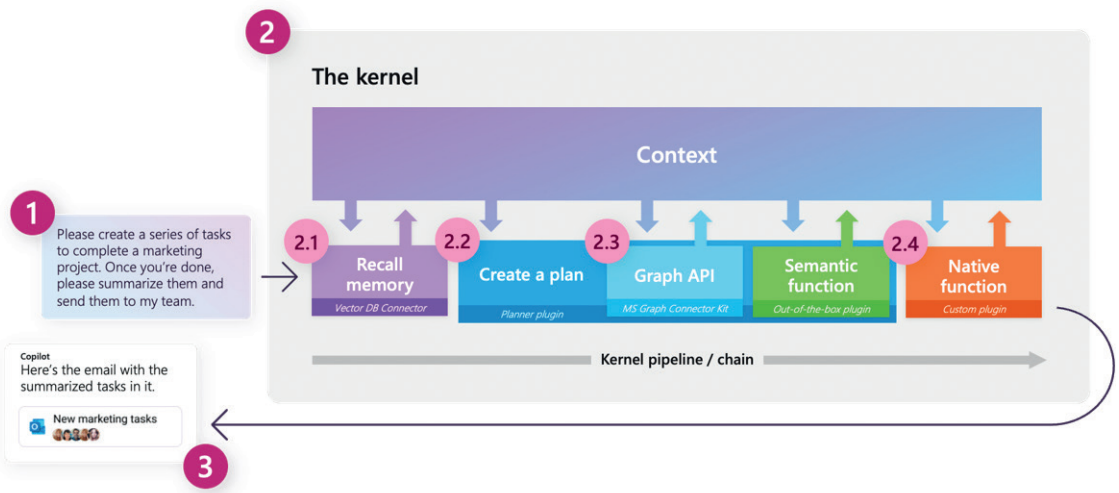
> Be data-aware: connect a language model to other sources of data
> Be agentic: Allow a language model to interact with its environment

```
using Azure;
using Azure.AI.OpenAI;
using static System.Environment;

string endpoint = GetEnvironmentVariable("AZURE_OPENAI_ENDPOINT");
string key = GetEnvironmentVariable("AZURE_OPENAI_KEY");

// Enter the deployment name you chose when you deployed the model.
string engine = "gpt-35-turbo-instruct";

OpenAIClient client = new(new Uri(endpoint), new AzureKeyCredential(key));

string prompt = "When was Microsoft founded?";
Console.Write($"Input: {prompt}\n");

Response<Completions> completionsResponse =
    await client.GetCompletionsAsync(engine, prompt);
string completion = completionsResponse.Value.Choices[0].Text;
Console.WriteLine($"Chatbot: {completion}");
```

1

> Vind deze en de andere listingen uit het artikel op sdn.nl!

As such, the LangChain framework is designed with the objective in mind to enable those types of applications. Deze quote geeft goed weer waaraan gedacht moet worden bij het implementeren van een LLM binnen jouw applicatie en waarom een framework zoals LangChain of Semantic Kernel gebruikt dient te worden.
Beide frameworks zijn prima oplossingen, maar als lezer van dit magazine is het waarschijnlijk een goed idee om voor Semantic Kernel te kiezen. De reden? Semantic Kernel heeft ondersteuning voor C#, waar LangChain dat op dit moment niet heeft.
Door het gebruik van Semantic Kernel kunnen flows van applicatie logica worden opgebouwd (plans) en automatisch worden uitgevoerd. Deze plans worden door een LLM opgesteld op basis van documentatie van de applicatie logica. Uiteraard kunnen ook reguliere prompts worden gebruikt voor het maken van complexe output. Een voorbeeld hoe Semantic Kernel te werk gaat.



Stap 1 is de input leveren, in dit geval een vraag om iets uit te voeren. In stap 2 worden verschillende onderdelen uitgesplitst, zoals het opvragen van de conversatie (2.1), een plan maken (2.2) op basis van beschikbare connectoren en semantic functions (2.3) en native functions (2.4). Zodra het plan is gemaakt en uitgevoerd kan de output worden geretourneerd naar de gebruiker (3).

Klinkt goed, hoe ga ik het dan gebruiken?

Eigenlijk kan iedereen direct starten met Semantic Kernel te gebruiken, het enige dat noodzakelijk is, is het toevoegen van een Azure Open AI instantie met een model. Er kan ook met de publieke Open AI instantie worden gewerkt, mocht je dat graag willen.
Zodra deze instantie bestaat kunnen de gegevens worden gebruikt voor het instantiëren van een `IKernel` object, wat als basis dient voor het werken met Semantic Kernel. Ten tijde van schrijven is net de v1.0.0 Beta1 SDK uitgebracht. Het framework staat nog in de kinderschoenen, maar het lijkt er naar op dat er snel een stabiele versie beschikbaar komt.
Het maken van een `IKernel` object is een relatief zware operatie, waardoor er over moet worden nagedacht welke lifetime dit object moet gaan krijgen binnen je applicatie. Een voorbeeld uit een van m'n eigen samples.

Listing 2

Na registratie kan het object worden gebruikt om een plan te maken.

Er zijn verschillende planners beschikbaar in Semantic Kernel, BasicPlanner, ActionPlanner, SequentialPlanner en StepwisePlanner. Alle vier hebben een ander doel. Wanneer je net start is het leuk om te beginnen met een `ActionPlanner`, deze maakt een plan met een enkele stap. De echte waarde zit hem echter in de `SequentialPlanner` en `StepwisePlanner`. De `SequentialPlanner` is namelijk in staat om van een vraag

BIO

Jan de Vries

Jan werkt als een software engineer aan Microsoft Azure. Op dit moment werkt hij in het team van Microsoft Cloud for Manufacturing aan een grote schaalbare oplossing binnen dat domein.
In de jaren hiervoor heeft hij als consultant vele rollen gehad met focus op het ontwerpen en ontwikkelen van oplossingen voor klanten binnen Microsoft Azure.
Mocht je meer van hem willen lezen, dan is het aan te raden om hem te volgen op zijn blog (https://jan-v.nl) of een van de social media kanalen.

```
s.AddSingleton(
    typeof(IKernel),
    s =>
    {
        var kernel = new KernelBuilder()
            .WithAzureTextCompletionService(
                        openAiSettings.ServiceModelName,
                        openAiSettings.ServiceCompletionEndpoint,
                        openAiSettings.ServiceKey
            )
            .WithAzureTextEmbeddingGenerationService(
                        openAiSettings.EmbeddingsDeploymentId,
                        openAiSettings.ServiceCompletionEndpoint,
                        openAiSettings.ServiceKey
            )
        .Build();
```

2

```
Microsoft.SemanticKernel.Planning.Stepwise.StepwisePlannerConfig config = new()
{
    MaxTokens = 2000,
    MaxIterations = 2,
};
var planner = new StepwisePlanner(kernel, config);
var plan = planner.CreatePlan(request);

Microsoft.SemanticKernel.AI.TextCompletion.CompleteRequestSettings settings = new()
{
    MaxTokens = 2000,
};
return await plan.InvokeAsync(kernel.CreateNewContext(), settings: settings);
```

3

(commando) een plan te maken van een of meerdere stappen die moeten worden uitgevoerd op basis van de informatie en beschikbare connectoren en plugins die beschikbaar zijn. De `StepwisePlanner` is een evolutie van de `SequentialPlanner`. In essentie zijn ze vergelijkbaar, maar de `StepwisePlanner` heeft als voordeel dat het ook kan leren. Wanneer het een plan heeft opgesteld om stap 1, 2, 3 uit te voeren en tijdens het uitvoeren er achter komt dat stap 2 niet de gewenste output geeft, dan wordt er een nieuwe plan opgesteld en kan er een alternatief plan worden opgesteld om stap 1, 4, 5, 3 uit te voeren om het commando succesvol te beantwoorden. De planner kan dit oneindig doen, dus het is aan te raden om een limiet op het aantal retries te zetten.
Bij deze een voorbeeld hoe de `SequentialPlanner` gebruikt kan worden. **Listing 3**

Hoe voeg ik stappen toe aan het gemaakte plan?

Het antwoord op deze vraag is: Niet! Het is niet de bedoeling om zelf aan het plan te gaan sleutelen en daarna uit te gaan voeren. Mocht je zelf stappen toegevoegd willen zien, om zo bijvoorbeeld business logica toe te voegen, bedrijfsprocessen te activeren, of iets totaal anders, dan moeten er Functions worden gemaakt.

Hoe maak ik mijn eigen Plugins of Functions?

Er zijn momenteel twee type Functions, namelijk Semantic Functions en Native Functions. Alle type Functions worden gebundeld in Plugins. Eerder werden deze Plugins nog Skills genoemd. Het komt dus nog voor dat in de documentatie en API van Semantic Kernel nog her en der de term Skills voorbijkomt, dit zal in de loop der tijd veranderen.

Semantic Functions

Een Semantic Function is niets anders als een prompt. Bij deze een voorbeeld voor het samenvatten van een HTML-pagina. **Listing 4**
Bij een dergelijke prompt, welke wordt bewaard in een `skprompt.txt` bestand, dient een `config.json` bestand met de configuratie voor de Semantic Function te worden toegevoegd. Deze configuratie bevat omschrijvingen van de input parameters, het doel van de Function en enkele andere parameters om bijvoorbeeld de temperatuur van het antwoord te tunen. Een voorbeeld van de `config.json` voor bovenstaande prompt. **Listing 5**
Door een correcte omschrijving toe te voegen is Semantic Kernel (en onderwater het LLM) in staat om een plan op te stellen met de beste Functions welke gebruikt kunnen worden om een antwoord op een vraag te verkrijgen.

Native Functions

Native Functions zullen de meeste lezers aanspreken. Het stelt je

```
The content is in the below block.
###
{{$input}}
###
Ignore all of the HTML markup and focus on the actual content.
Summarize the actual content.
```

4

```
{
  "schema": 1,
  "type": "completion",
  "description": "Create a summary from the specified HTML content.",
  "completion": {
    "max_tokens": 2048,
    "temperature": 0.3,
    "top_p": 0.0,
    "presence_penalty": 0.0,
    "frequency_penalty": 0.0
  },
  "input": {
    "parameters": [
      {
        "name": "input",
        "description": "The HTML content that needs to be summarized.",
        "defaultValue": ""
      }
    ]
  }
}
```

5

namelijk in staat om de bestaande codebase te ontsluiten in een Plan. Om bestaande code te ontsluiten dienen er enkele attributen te worden toegevoegd op de entry-points van de logica. Op moment van schrijven zijn dit de attributen `SKFunction` en een `Description`. De eerste geeft aan dat het hier gaan om een Native Function welke door Semantic Kernel gebruikt kan worden. De tweede wordt door Semantic Kernel gebruikt om te bepalen of een Function gebruikt moet worden om een vraag/commando te beantwoorden. Om in het domein van de vorige voorbeelden te blijven, hier een voorbeeld Native Function voor het downloaden van de HTML van een website. **Listing 6**
Wanneer alle, of veel, logica binnen jouw applicatie wordt ontsloten als een Native Function, dan kunnen complexe flows eenvoudig worden opgesteld en uitgevoerd. Dit is iets dat wij binnen ons project veel gebruiken.

```
[SKFunction, Description("Retrieve the body from a specified website URL.")]
public async Task<string> GetBody(string websiteUrl)
{
    logger.LogDebug("Retrieving the content for `{websiteUrl}`.", websiteUrl);
    var content = await GetContent(websiteUrl);

    return content;
}
```

6

```
void AddSemanticSkills()
{
    logger.LogInformation("Importing semantic skills");
    string skillsPath = Path.Combine(Path.GetDirectoryName(Assembly.GetExecutingAs-
sembly().Location)!, "GenerativeAi", "SemanticSkills");
    kernel.ImportSemanticSkillFromDirectory(skillsPath, "Domain");
}

void AddNativeSkills()
{
    logger.LogInformation("Importing native skills");
    kernel.ImportSkill(new DownloadContent(s.GetRequiredService<IHttpClientFacto-
ry>(), s.GetRequiredService<ILogger<DownloadContent>>()), "MySkills");
}
```

7

De Functions toevoegen

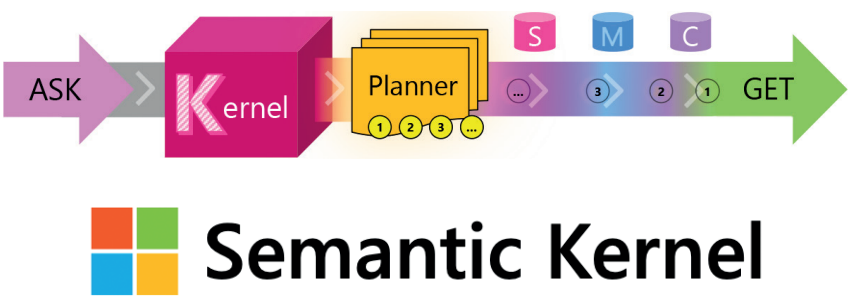
Semantic Kernel komt standaard met ingebouwde Plugins en bijbehorende Functions. Het toevoegen van jouw eigen Plugins is ook mogelijk door deze aan het `IKernel` object toe te voegen.
Zie hier een voorbeeld van bovenstaande Functions. **Listing 7**

Conclusie, moet ik altijd Semantic Kernel gebruiken?

Het korte antwoord hierop is: Nee!
Het gebruik maken van Semantic Kernel en een LLM biedt veel voordelen voor het maken van complexe flows van logica. Wij maken hier heel veel gebruik van, omdat we van tevoren niet alle flows binnen de applicatie kunnen definiëren. En al zouden we alle flows kunnen definiëren, dan zou dat resulteren in een enorm complexe decision tree. In ons geval biedt Semantic Kernel en de onderliggende LLM dus heel veel voordelen.

Met Semantic Kernel kunnen flows van logica worden opgebouwd en automatisch uitgevoerd.

Zo zal er per situatie beoordeeld moeten worden of het echt voordelen biedt om een vraagstuk van de business (gebruikers) te beantwoorden. Wel adviseer ik om deze frameworks goed in de gaten te houden, want mocht het momenteel nog geen direct nut hebben om te gebruiken binnen jouw applicatie, dan is dit nut er over een tijdje misschien wel. En zoals eerder geschreven, stelt dit je in staat om complexe vraagstukken op een andere manier te implementeren welke mogelijk tot een eenvoudigere codebase resulteert.
De voorbeelden uit dit artikel zijn geplukt uit de volgende GitHub repository:
https://github.com/Jandev/sk-plugin-sample
Deze repository bevat enkele voorbeelden voor het maken van plugins en het toevoegen van een publieke Open AI plugin binnen je eigen applicatie om zo extra functionaliteit toe te voegen aan een applicatie. ●



Managers

Vanaf jongs af aan heb ik geen positief beeld over managers gehad. Het woord alleen al werd thuis met walging uitgesproken. Managers zijn mensen die jou vertellen wat je moet doen, terwijl ze geen flauw benul hebben wat er op de werkvloer speelt. Ze zijn vooral goed in praten, doen zelf niets, en laten het echte werk aan andere mensen over.

Auteur: Nadine Wolf

Toen ik zelf carrière ging maken en mij ging verdiepen in leidinggeven werd mijn beeld van managers er niet beter op. Bij veel van mijn studiemateriaal over modern leiderschap (denk bijvoorbeeld aan een Simon Sinek) wordt er onderscheid gemaakt tussen managers en leidinggevenden, waarbij managers het onderspit delven. Het begrip manager gebruik ik regelmatig om een collega te omschrijven. In eerste instantie niet in positieve zin. Vaak in de trant van iemand die het niet snapt; iemand die nog in de vorige eeuw leeft wat betreft werkstijl.
Ondanks mijn negatief beeld over managers heb ik ambities om zelf een manager te worden. Niet omdat ik mijzelf als wereldverbeteraar zie en "die managers" wil laten zien hoe het wel moet. Nee, ik wil meer invloed uitoefenen op de projecten waaraan ik werk.
Het grappige is dat ik die managers wel begin te waarderen: ze zijn niet volledig nutteloos. Ik vind het wel fijn als er iemand is die het een en ander voor je regelt zodat ik kan concentreren op "het echte werk". Daarnaast oefenen managers stiekem best wel wat invloed uit op het "hogere management": de CEO's en directeuren. Je weet wel; die mensen die bepalen welke richting een bedrijf heengaat, waarin geïnvesteerd wordt en waarin niet, en welke projecten voorrang krijgen.
Vooral in de afgelopen paar jaar ben ik erachter gekomen dat je iets kan leren van die managers van de vorige eeuw. Je kan dan misschien moeite hebben met hun "oubollige" manier van mensen aansturen, maar vergeet niet dat ze al jarenlang werkervaring hebben opgedaan! Managers delen graag hun geleerde lessen met je zodra je interesse toont. Het maakt niet uit welke generatie het is, mensen blijven mensen. Managers hebben veel ervaring in hoe je met mensen, en vooral andere managers, omgaat. Ze weten hoe de politieke spelletjes gespeeld worden en zijn experts in diplomatiek. Iets waar je als software engineer het liefs ver van weg wilt blijven totdat je erachter komt dat juist deze dingen je dagelijkse werkzaamheden beïnvloeden.
Ze zeggen niet voor niets 'if you can't beat them, join them'. Je kan ze als je grootste vijand of obstakel zien, en met een hoop frustratie proberen te overtuigen dat we toch echt over moeten gaan naar een nieuw platform. Of vraag ze om hulp en leer van ze. Vraag ze hoe zij het zouden aanpakken wanneer ze in jouw schoenen staan. Kortom: maak ze je bondgenoot en een wereld gaat voor je open. ●

CHATGPT **en C#**

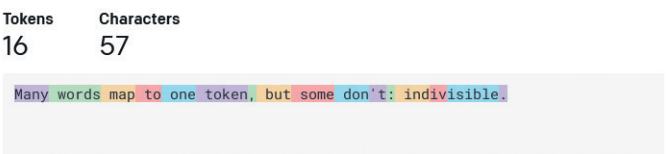
Dit artikel beschrijft een oplossing in C# om die het mogelijk maakt om met natuurlijke taal vragen stellen over een PDF door gebruik te maken van de Chat- en Embeddings-functionaliteit van ChatGPT gecombineerd met de vector zoekmogelijkheden van Redis.

Auteur: Stef Heyenrath

Uitdaging

Copy-Paste

Het simpelweg kopiëren van de tekst uit een PDF en deze tekst aanbieden aan ChatGPT zal niet werken omdat de OpenAI Chat Completion (voor de meeste modellen) een maximale tokenlengte van 4096 heeft.
In de meeste talen waaronder Engels, is de verhouding van tokens tot woorden ongeveer ¾, wat resulteert in bijna 3000 woorden die gebruikt kunnen worden in OpenAI Chat Completion.
Zie bijvoorbeeld onderstaande Engelse tekst die bestaat uit 10 woorden wat equivalent is met 16 tokens omdat ook leestekens meetellen en sommige woorden bestaan uit meerdere tokens.



Aanpak

Om deze beperking te omzeilen, moet een andere aanpak worden gekozen:

> Splits het PDF-document in meerdere tekstfragmenten.
> Embed elk tekstfragment met behulp van OpenAI.
> Sla deze embeddings op in een vector-database zoals Redis of Pinecone.
> Bij het stellen van een vraag, embed deze vraag en gebruik de vector zoekmogelijkheid van de vector-database om relevante tekstfragmenten terug te geven.
- Wanneer 1 of meer tekstfragmenten zijn gevonden, gebruik ChatGPT om de vraag te beantwoorden op basis van deze tekstfragmenten.

Alle bovenstaande stappen worden in dit artikel in detail beschreven.

Overzicht

Dit artikel is verdeeld in de volgende hoofdstukken:

> Hoofdstuk 1 beschrijft de wiskundige achtergrond voor "Vectorisatie" en "Cosine Similarity".
> Hoofdstuk 2 beschrijft hoe toegang te krijgen tot OpenAI in C# en hoe de Chat, Moderation en Embeddings API aan te roepen.
> Hoofdstuk 3 beschrijft hoe een PDF-document te lezen en de tekst in tekstfragmenten te splitsen.
> Hoofdstuk 4 beschrijft hoe toegang te krijgen tot Redis in C# en hoe embeddings op te slaan en te bevragen.
> Hoofdstuk 5 beschrijft de complete werkende oplossing in C# als demo.

1. Wat zijn Vectorisatie en Cosine Similarity?

1.1 Vectorisatie

Vectorisatie is het proces waarbij woorden of zinnen worden omgezet in numerieke vectoren waarbij er twee opties zijn:

> **Word** embeddings: die individuele woorden omzetten in vectoren (bijvoorbeeld "Word2Vec" en "GloVe"). Deze worden gegenereerd via neurale netwerken die zijn getraind op grote teksten om semantische en syntactische relaties tussen woorden te vangen en te creëren.
> **Document** embeddings: die hele documenten representeren als vectoren (bijvoorbeeld "Doc2Vec").

Om de achterliggende wiskunde wat meer te verduidelijken nemen we twee zinnen als voorbeeld:
A. "The cat sits on the mat."
B. "The dog sits on the rug."
We kunnen deze twee zinnen omzetten in numerieke vectoren met behulp van word embeddings. Laten we voor de eenvoud aannemen dat we een 3-dimensionale word embedding-space (elk woord is dan een vector met 3 cijfers) hebben en dummy-waardes gebruiken:

> the: [0.1, 0.2, 0.3]
> cat: [0.4, 0.5, 0.6]
> dog: [0.45, 0.55, 0.65]
> sits: [0.05, 0.15, 0.25]
> on: [0.2, 0.3, 0.4]
> mat: [0.6, 0.7, 0.8]
> rug: [0.65, 0.75, 0.85]

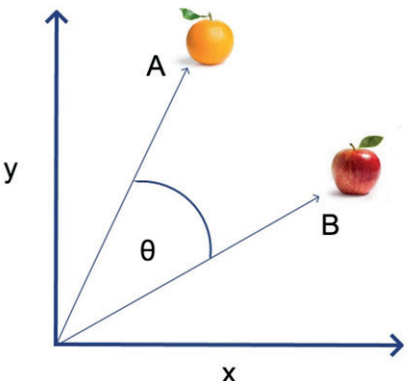
OpenAI gebruikt een veel grotere (1536-dimensionale) word embedding-space, waardoor de semantische en syntactische samenhang tussen de stukken tekst veel uitgebreider, complexer en beter is.
Een manier om elke zin (zin A en zin B) als een vector te representeren, is door het gemiddelde van de word embeddings te nemen (alleen ter illustratie en de uitleg is hiervoor gekozen):

> A vector: [(0.1+0.4+0.05+0.2+0.6)/5, (0.2+0.5+0.15+0.3+0.7)/5,

(0.3+0.6+0.25+0.4+0.8)/5] = [0.27, 0.37, 0.47]
> B vector: [(0.1+0.45+0.05+0.2+0.65)/5, (0.2+0.55+0.15+0.3+0.75)/5, (0.3+0.65+0.25+0.4+0.85)/5] = [0.29, 0.39, 0.49]

1.2 Cosine Similarity

"Cosine Similarity" is een metriek die wordt gebruikt om de gelijkenis tussen twee vectoren te bepalen. Het berekent de cosinus van de hoek tussen deze vectoren, met een waarde variërend van -1 (volledig verschillend) tot 1 (identiek).



In de context van NLP (Natural Language Processing), wordt "Cosine Similarity" gebruikt om de gelijkenis tussen teksten te meten.
De formule is als volgt gedefinieerd:

$$\text{cosine similarity} = S_C(A,B) := \cos(\theta) = \frac{\mathbf{A}\cdot\mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

1.2.1 Wiskundig voorbeeld voor Cosine Similarity

Met behulp van de zin-vectoren berekend in het vorige voorbeeld:
> Zin A vector: [0.27, 0.37, 0.47]
> Zin B vector: [0.29, 0.39, 0.49]
Kunnen we de Cosine Similarity berekenen tussen deze twee vectoren met behulp van de volgende tussen stappen:
> Het inproduct (of dot product) van de twee vectoren is 0.4529
> De grootte van deze twee vectoren zijn: A. 0.6563 B. 0.6901
> De Cosine Similarity = 0.4529 / (0.6563 * 0.6901) = 0.9999416 =˜ 1

De Cosine Similarity tussen de twee zin-vectoren is ongeveer 1, wat duidt op een hoge mate van gelijkenis.
In dit vereenvoudigde geval worden de zinnen als identiek beschouwd, ondanks het gebruik van verschillende woorden (cat/dog en mat/rug), omdat de structuur en betekenis zeer vergelijkbaar zijn.
Let op: dit resultaat is slechts een heel eenvoudig voorbeeld en gebruikt het gemiddelde van de word embeddings. In de praktijk worden geavanceerde methoden gebruikt zoals het HNSW (Hierarchical Navigable Small World) algoritme.

1.3 Voorbeelden

1.3.1 Voorbeeld 1: Filmaanbevelingen

Stel je voor dat je net een fantastische film hebt gekeken en vergelijkbare films wilt vinden om van te genieten. Filmaanbevelingen, zoals die op streamingplatforms, gebruiken Vectorized Search en Cosine Similarity om films met vergelijkbare verhaallijnen, thema's of genres te vinden.
Het systeem zet eerst filmbeschrijvingen, trefwoorden of andere tekstuele gegevens om in vectoren met behulp van technieken die word embeddings gebruiken. Vervolgens berekent het de Cosine Similarity tussen de vectoren van de film die je net hebt gekeken en de vectoren van andere films in de database. Een hogere score betekent dat deze films vergelijkbaar zijn.

1.3.2 Voorbeeld 2: Klantenservice

Klantenservicecentra ontvangen vaak grote hoeveelheden vragen die veel tijd en moeite kosten om te beantwoorden. Door Vectorized Search en Cosine Similarity te gebruiken, kunnen support medewerkers snel vergelijkbare eerdere vragen (bijvoorbeeld opgeslagen in een database) en hun bijbehorende oplossingen identificeren.
De vraag van de klant en de bestaande database met vragen en oplossingen worden omgezet in vectoren. En net als in het vorige voorbeeld wordt de score gebruikt om snel en correct de vragen van de gebruikers te beantwoorden.

2. Toegang tot OpenAI en het aanroepen van de Chat, Moderation en Embeddings API

2.1 OpenAI API

OpenAI biedt een REST API waarmee ontwikkelaars toegang krijgen tot alle functionaliteiten van het OpenAI-systeem.

2.2 .NET Client-bibliotheken

Er zijn verschillende .NET API Client-bibliotheken beschikbaar, voor mijn oplossing heb ik voor het NuGet OpenAI gekozen. Deze bibliotheek ondersteunt o.a. de volgende API's:
> Chat: Tekstgeneratie in de vorm van chatberichten.
> Moderation: Classificeer tekst tegen het OpenAI contentbeleid.
> Completions: Tekstgeneratie API gebruikt prompts om taken over samenvatting, vertaling, chatbots en meer te voltooien met eenvoudige instructies.
> Embeddings: Transformeer tekst in een vector (lijst) van zwevende kommagetallen.

2.3 Toegang tot de OpenAI API

Voordat je de API van OpenAI kunt gebruiken, moet je een API-key genereren. Voer de volgende stappen uit:
- Ga naar https://platform.openai.com/account/api-keys
- Klik op de knop 'Create new secret key' en schrijf de gegenereerde

sleutel op, want deze wordt slechts één keer weergegeven. Je hebt deze sleutel nodig om toegang te krijgen tot de OpenAI API.
- Onthoud ook je 'Organization ID' die je hier kunt vinden: https://platform.openai.com/account/org-settings

2.4 Gebruik van de client API om de API aan te roepen

Maak eerst een instantie van de API door middel van Dependency Injection: **Voorbeeld 1**
Zie de stappen hieronder over hoe je elke API aanroept met behulp van C#.

2.4.1 Aanroepen van de Moderation API

De complete oplossing moet gebruikers in staat stellen om vragen te stellen over het PDF-document, maar we willen een beschermingslaag toevoegen zodat schadelijke of ongeldige vragen worden afgewezen. Hiervoor kan de Moderation API van OpenAI worden gebruikt (het aanroepen van deze API is gratis).
Het aanroepen van de Moderation gebeurt als volgt en afhankelijk van de uitkomst (elk resultaat wordt door OpenAI gemarkeerd als ongeldig of schadelijk), wordt de vraag afgewezen: **Voorbeeld 2**

2.4.2 Aanroepen van de Chat API

Dit is equavalent met een gesprek in Chat API: **Voorbeeld 3**
In C# gaat dit als volgt: **Voorbeeld 4**

2.4.3 Aanroepen van de Embeddings API **Voorbeeld 5**

3 Een PDF-document lezen en de tekst opdelen in tekstfragmenten

3.1 Doel

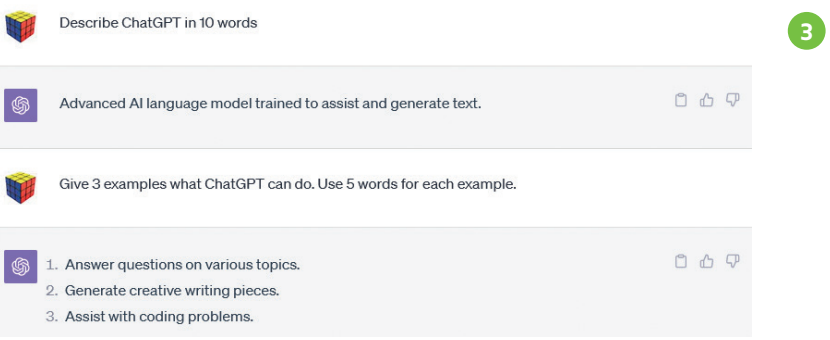
Het doel van de oplossing is om een PDF-document te lezen en te verwerken en de volledige tekst op te splitsen in meerdere tekstfragmenten om het Embedden mogelijk te maken.

```
public class OpenApiCode
{

    static async Task Main(string[] args)
    {
        var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices(services =>
            {
                services.AddSingleton<IOpenAIAPI>(_ => new OpenAIAPI("Your OpenAI API-Key"));
            })
            .Build();

        var api = host.Services.GetRequiredService<IOpenAIAPI>();
```

1

```
// Call the Chat API to do a moderation on a text
ModerationResult moderation = await api.Moderation.CallModerationAsync("My question");
if (moderation.Results.Any(r => r.Flagged))
{
    Console.WriteLine("Sorry, that question is not allowed.");
}
```

2



3

```
// Call the Chat API to create an conversation
Conversation chat = api.Chat.CreateConversation();

// Add user input
chat.AppendUserInput("Describe ChatGPT in 10 words");

// Get the response
string response = await chat.GetResponseFromChatbotAsync();
```

4

```
// Call the API to embed text using the default embedding model
float[] embeddings1 = await api.Embeddings.GetEmbeddingsAsync("Hello World");
```

5

3.2 NuGet-package

Er zijn verschillende NuGet-packages beschikbaar voor het lezen van een PDF, voor deze oplossing heb ik gekozen voor PdfPig.

3.3 Tekst opsplitsen

Omdat het niet mogelijk is om de volledige PDF-tekst in ChatGPT te voeren, moet deze grote tekst worden opgesplitst in meerdere tekstfragmenten. Mijn aanpak hier is om alle tekst uit de PDF te lezen en deze tekst op te splitsen in regels (splits-tekens zijn: "", "\r" en "\n"). Als de regel zelf meer dan 1000 tekens bevat, wordt deze regel op een woordgrens gesplitst. De volgende stap is opdelen in brokken (met behulp van 1000 tekens) met volledige regels.

3.4 Code

Dit hoofdgedeelte van de code ziet er als volgt uit: **Voorbeeld 6**

4. Toegang tot Redis in C# en het invoegen en zoeken van embeddings

4.1 UI

Voor het bekijken van de Redis--database en het uitvoeren van CLI-commando's op Redis, gebruik ik RedisInsight.

```
static IReadOnlyList<string> Split(string filePath)
{
    var stringBuilder = new StringBuilder();

    using (var document = PdfDocument.Open(filePath))
    {
        foreach (var page in document.GetPages().Where(p => !string.IsNullOrWhiteSpace(p.Text)))
        {
            stringBuilder.Append(page.Text.Trim());
        }
    }

    string[] lines = SplitToLines(stringBuilder);

    return GetTextFragments(lines)
        .Where(line => !string.IsNullOrWhiteSpace(line))
        .ToArray();
}
```

6

4.2 Redis SDK's

Er zijn meerdere .NET NuGet-packages beschikbaar, voor deze oplossing heb ik gekozen voor:

- StackExchange.Redis voor het invoegen van gegevens en het creëren van indexen
- NRedisStack voor het zoeken van gegevens

4.3 Tekstfragmenten opslaan met behulp van Redis hashes

We zullen verschillende Redis hashes creëren, één voor elk tekstfragment en deze worden gegroepeerd voor een PDF-document met behulp van een voorvoegsel.
Hashes zijn records gestructureerd als verzamelingen van key-value pairs.
Elke hash heeft de volgende velden:

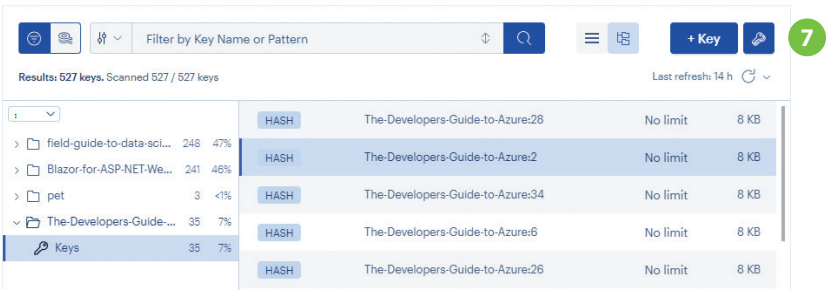
- **idx**: De 0-gebaseerde index van het tekstfragment.
- **text**: Het daadwerkelijke tekstfragment, dit is vereist wanneer de relevante tekstfragmenten worden gevoed in de ChatGPT API om de vraag van de gebruiker te beantwoorden.
- **tokens**: Het aantal tokens dat nodig is om het tekstfragment op te slaan, nodig om ervoor te zorgen dat de totale lengte van het relevante tekstfragment de tokenlengte van 4096 niet overschrijdt.
- **embedding**: De embedding van het tekstfragment (de vector), gecreëerd met de OpenAI embeddings API. Deze vector wordt opgeslagen als een reeks bytes.

In RedisInsight worden de PDF-documenten weergegeven in een TreeView: **Voorbeeld 7**
De TreeView-optie is ingeschakeld omdat alle hashes zijn voorafgegaan door de naam van het PDF-document ("The-Developers-Guide-to-Azure"). De hash met index 214 van een PDF-document The-Developers-Guide-to-Azure ziet er als volgt uit: **Voorbeeld 8**
Om de tekstfragmenten en embeddings van berichten op te slaan, kunnen we de volgende C#-code gebruiken:

4.3.1 Registreer een eigen Redis-DatabaseService met behulp van DI: **Voorbeeld 9**

4.3.2 Gegevens invoegen in Redis

Gebruik de onderstaande C#-code om een enkele hash-entry in Redis in te voegen: **Voorbeeld 10**



7

8

```
static async Task Main(string[] args)
{

    using var host = Host.CreateDefaultBuilder(args)
        .ConfigureServices(services =>
        {
            services.AddSingleton<IConnectionMultiplexer>(_ =>
            {
                var options = new ConfigurationOptions { EndPoints = { "localhost:6380" } };
                return ConnectionMultiplexer.Connect(options);
            });
        })
        .Build();

    // Get IConnectionMultiplexer
    var connection = host.Services.GetRequiredService<IConnectionMultiplexer>();

    // Get an interactive connection to a database inside Redis.
    var database = connection.GetDatabase();
```

9

```
// Get an interactive connection to a database inside Redis.
var database = connection.GetDatabase();

// The prefix used to store the hashes
string prefix = "The-Developers-Guide-to-Azure";

// The index from the text-fragment
int idx = 214;

// The text-fragment
string text = "Azure Defender enables extended ...";

// The number of tokens for the text-fragment (calculated using SharpToken)
int tokens = 175;

// The OpenAI Embeddings API is used to generate the vectors
float[] embeddings = new float[] { 1.0f };

// Create a byte array from embeddings
byte[] byteArray = MemoryMarshal.Cast<float, byte>(embeddings).ToArray();

// Insert the Hash-entry with the index, text-fragment, number of tokens and the embedding
database.HashSet($"{prefix}:{idx}", new HashEntry[]
{
    new(new RedisValue("idx"), idx),
    new(new RedisValue("text"), text),
    new(new RedisValue("tokens"), tokens),
    new(new RedisValue("embedding"), byteArray)
});
```

10

```
static async Task CreateIndex(IDatabase database)
{
    string indexName = "The-Developers-Guide-to-Azure-index"; // The name of the index
    string prefix = "The-Developers-Guide-to-Azure"; // The prefix used to store the hashes

    var schema = new Schema()
        .AddNumericField("idx")
        .AddVectorField("embedding",
            Schema.VectorField.VectorAlgo.HNSW,
            new Dictionary<string, object>
            {
                { "TYPE", "FLOAT32" },
                { "DIM", "1536" },
                { "DISTANCE_METRIC", "COSINE" }
            }
        );

    var createParams = new FTCreateParams()
        .AddPrefix($"{prefix}:");

    await database.FT().CreateAsync(indexName, createParams, schema);
}
```

11

De tekstfragmenten zijn nu opgeslagen als Redis hashes.

4.4 Indexen

De RediSearch-functionaliteit wordt gebruikt om een invoervraag te matchen met één of meer van deze tekstfragmenten. RediSearch ondersteunt vector similarity semantic search. Voor zo'n zoekindex om te werken, moet er een index worden gecreëerd die dat vectorveld ondersteunt.

4.4.1 Creëer een index in Redis

NRedisStack wordt gebruikt om de index in Redis te creëren. **Voorbeeld 11**
Bij het creëren van een index, definieer de velden om te indexeren op basis van een schema. In het bovenstaande codevoorbeeld wordt het indexveld (idx) toegevoegd als een numeriek veld en het vectorveld (embedding) wordt gedefinieerd als een VectorField.
De VectorField-class wordt gebruikt om het vectorveld te maken en heeft de volgende eigenschappen:

- Naam: de naam van het veld (embedding)
- Algoritme: FLAT of HNSW kan hier worden gedefinieerd: FLAT: wanneer zoekkwaliteit van hoge prioriteit is en zoeksnelheid minder belangrijk is en HNSW: geeft snellere bewerking.
- Attributen: een dictionary dat het volgende specificeert: het datatype (float), het aantal dimensies van de vector (1536 voor text-embedding-ada-002 van OpenAI)- en de afstandsmaat (COSINE) voor Cosine Similarity, wat wordt aanbevolen door OpenAI met hun embedding-model.

Met RedisInsight kun je de indexen bekijken: **Voorbeeld 12**

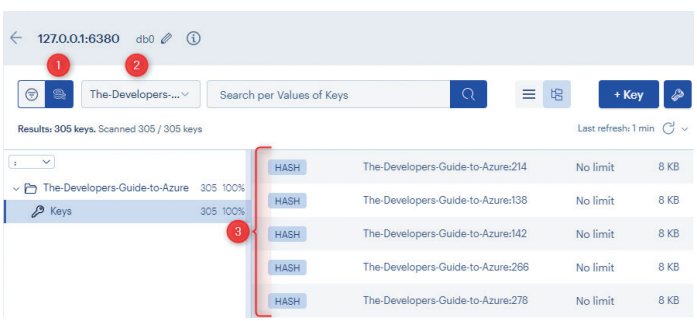
- 1. Schakel over naar de index-weergave
- 2. Dit is de naam van de index ("The-Developers-Guide-to-Azure-index")
- 3. De hashes

4.5 Een Cosine Similarity zoekopdracht uitvoeren in Redis

Met de gecreëerde hashes en index kunnen we nu een Cosine Similarity zoekopdracht uitvoeren op basis van de vraag (met behulp van natuurlijke taal) van de gebruiker.
Redis wordt gebruikt om tekstfragmenten te vinden die vergelijkbaar zijn met de queryvector.

4.5.1 Converteer de vraag naar een vector

Gebruik de OpenAI Embeddings API om de vraag van de gebruiker om



12

```
static async Task<byte[]> ConvertTheQuestionToAVector(IOpenAIAPI api)
{
    string question = "What is Azure DevOps?";
    float[] questionAsVector = await api.Embeddings.GetEmbeddingsAsync(question);
    byte[] questionAsBytes = MemoryMarshal.Cast<float, byte>(questionAsVector).ToArray();

    return questionAsBytes;
}
```
13

```
static Query BuildQuery(byte[] questionAsBytes)
{
    var query = new Query("*=>[KNN 5 @embedding $vector AS vector_score]")
        .AddParam("vector", questionAsBytes)
        .ReturnFields("text", "tokens", "vector_score")
        .SetSortBy("vector_score")
        .Dialect(2);

    return query;
}
```
14

```
static async Task<IReadOnlyList<VectorDocument>> SearchAsync(IDatabase database, string indexName, Query query)
{
    SearchResult searchResult = await database.FT().SearchAsync(indexName, query);

    return searchResult.Documents
        .Select(document => new VectorDocument
        (
            Idx: (int)document["idx"],
            Text: document["text"]!,
            TokenLength: (int)document["tokens"],
            Score: (float)document["vector_score"]
        ))
        .OrderByDescending(document => document.Score)
        .ToArray();
}
```
15

te zetten in een vector (en converteer die vector naar een byte-array): **Voorbeeld 13**

4.5.2 Bouw de zoekopdracht

Bouw nu de Redis Cosine Similarity zoekopdracht: **Voorbeeld 14**
Deze query bevat de volgende onderdelen:
De Query-class gebruikt een queryString aan als constructor parameter met als waarde: "*=>[KNN 5 @embedding $vector AS vector_score]". Dit is gewoon een string met het queryformaat dat Redis verwacht door te geven aan de Query. Redis wordt geïnstrueerd om de 5 dichtstbijzijnde buren van $vector in de embedding-velden (tekstfragmenten) van de hashes te retourneren.
De @ wordt gebruikt om het embedding-veld te definiëren en $ om de vector aan te duiden die zal worden doorgegeven met behulp van de AddParam()-methode. Die vector is onze gevectoriseerde querystring. Met "AS vector_score" wordt een benoemde retourvariabele gegenereerd die kan worden gebruikt om de resultaten van hoog naar laag te rangschikken.
De ReturnFields methode definieert welke velden moeten worden geretourneerd door de Redis-zoekopdracht. In dit geval worden deze velden geretourneerd:

> text: Het tekstfragment als een string
> tokens: Dit definieert het aantal tokens voor dit tekstfragment
> vector_score: de score als een float

De SetSortBy methode wordt gebruikt om de resultaten te sorteren op vector_score.
Tevens is het vereist om het juiste dialect te specificeren. Het dialect is de versie van de querytaal. Versie 2 wordt hier gebruikt omdat dit overeenkomt met de query-syntaxis.

4.5.3 Het uitvoeren van de zoekopdracht

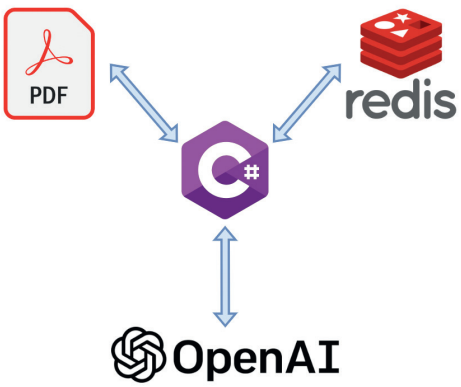
Het uitvoeren van de zoekopdracht in C# kan als volgt worden gedaan: **Voorbeeld 15**

Bovenstaande code voert een zoekopdracht uit op de gespecificeerde index. In de oproep naar de SearchAsync-methode worden de indexnaam en de eerder gebouwde query doorgegeven.
Toegang tot de vereiste velden kan gemakkelijk worden gedaan door de index-operator van het Document (de Document-class extends een dictionary) te gebruiken en gewoon de waarde naar het vereiste type te casten.
Het sorteren van het document op Score (vector_score) is vereist. De score varieert van -1,0 tot 1,0.

5 Complete oplossing

5.1 Overzicht van de oplossing

De onderstaande afbeelding toont het hoofd consoleprogramma in C# dat interactie heeft met een PDF-bestand, een Redis-database en de OpenAI API.



5.2 Classes

Het voorbeeld project bevat de volgende klassen:

5.2.1 DocumentSplitter

Deze klasse gebruikt PdfPig om een PDF-document te lezen en alle regels te extraheren, en zet deze om in tekstfragmenten met een maximale lengte van 1000 tekens.

5.2.2 RedisDatabaseService

Deze klasse bevat twee hoofd-functies:

> Data invoegen en een index creëren in Redis
> Een Cosine Similarity-zoekopdracht uitvoeren met Redis

Het toevoegen van alle tekstfragmenten op deze manier wordt gedaan:

5.2.3 MainService

Deze class combineert de logica van de DocumentSplitter en RedisDatabaseService om de OpenAI API aan te roepen om

> 1. Moderatie te doen (verificatie om ongeldige of schadelijke vragen uit te sluiten)
> 2. Embeddings te verkrijgen voor de vraag + tekstfragmenten
> 3. ChatCompletion te gebruiken om een correct antwoord te krijgen op basis van de gevonden documenten

Vervolgens is er logica om te controleren of er embeddings aanwezig zijn voor dit PDF-document in de Redis-database, als er geen gegevens zijn, wordt het PDF-document geanalyseerd en opgesplitst in tekstfragmenten met behulp van de DocumentSplitter-klasse en worden de tekstfragmenten + hun embeddings toegevoegd aan de Redis-database. Nu wordt een Cosine Similarity-zoekopdracht uitgevoerd die resulteert in 5 vector-documenten die de tekstfragmenten bevatten. Deze tekstfragmenten worden samengevoegd tot 1 string die wordt toegevoegd aan het ChatGPT-chatgesprek.
Dit chatgesprek bevat enkele gedetailleerde instructies en vereisten voor ChatGPT om de vraag correct te beantwoorden. De voorbeeld C#-code ziet er als volgt uit: **Voorbeeld 16**

Merk op dat er wat extra logica kan worden toegevoegd om ervoor te zorgen dat de maximale tokenlengte van het model niet wordt overschreden.

Een oplossing in C# om met natuurlijke taal vragen stellen door gebruik te maken van ChatGPT gecombineerd met Redis

In mijn oplossing neem ik alle 5 documenten (tekstfragmenten), je kunt ervoor kiezen om alleen de hoogste match of de hoogste 2 matches te nemen, dit hangt af van de behoefte.

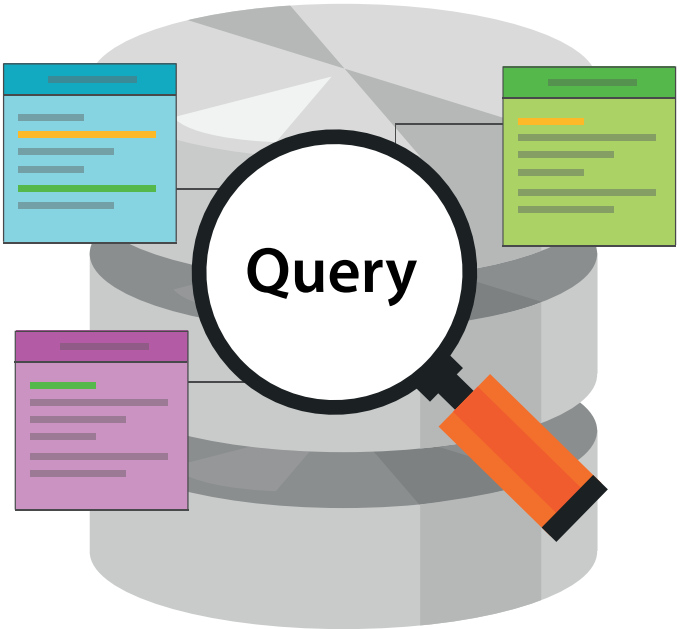
Conclusie

Dit artikel beschrijft hoe je een C#-oplossing bouwt met toegang tot de OpenAI API om embeddings te creëren en vragen te stellen aan ChatGPT op basis van inhoud uit de geïndexeerde PDF.
Een PDF indexeren is slechts een voorbeeld, deze oplossing kan namelijk ook prima toegepast worden op andere datasets zoals interne documenten of wiki-pagina's. ●

16

```
static async Task SearchForCosineSimilarityAndGetResponseFromChatGPTAsync(IDatabase database, IOpenAIAPI api)
{
    string indexName = "field-guide-to-data-science-index";
    string question = "What are fractals?";
    byte[] questionAsVector = await ConvertTheQuestionToAVectorAsync(api, question);
    Query query = BuildQuery(questionAsVector);

    // 5 VectorDocuments are returned
    IReadOnlyList<VectorDocument> vectorDocuments = await SearchAsync(database, indexName, query);

    // Just concatenate all text-fragments from the found documents.
    var textBuilder = new StringBuilder();
    foreach (var vectorDocument in vectorDocuments)
    {
        textBuilder.AppendLine(vectorDocument.Text);
    }

    // Here is where the 'magic' happens
    var contentBuilder = new StringBuilder();
    contentBuilder.AppendLine($"Based on the following source text \"{textBuilder.ToString()}\" follow the next requirements:");
    contentBuilder.AppendLine($"- Answer the question: \"{question}\"");
    contentBuilder.AppendLine(@"- Make sure to give a concrete answer");
    contentBuilder.AppendLine(@"- Do not start your answer with ""Based on the source text,""");
    contentBuilder.AppendLine(@"- Only base your answer on the source text");
    contentBuilder.AppendLine(@"- When you cannot give a good answer based on the source text, return ""I cannot find any relevant information.""");

    // Start a chat
    var chat = api.Chat.WithRetry(chatAPI => chatAPI.CreateConversation());

    // Add user input
    chat.AppendUserInput(contentBuilder.ToString());

    // Get the response for that question from ChatGPT.
    var response = await chat.WithRetry(conversation => conversation.GetResponseFromChatbotAsync());
}
```

Use LLMs to query SQL Database



Now with the availability of large language model capable of solving medium-complex problems, we can leverage its capabilities to generate SQL queries for our natural language asks. In this article, we will see various methods to accomplish this goal.

Auteur: Dhruv Patel

Prerequisites

There are some prerequisites you need to know, before starting to develop software using a LLM. Some of these basics are described below.

LLM Tokens

Tokens are the basic units of text in LLMs, which can range from one character to one word. For example, the sentence "Hello, world!" has 3 tokens: ["Hello", ",", " world!"].
Different LLMs have different token limits depending on their architecture and implementation. For instance, ChatGPT 3 has a token limit of 4096, GPT4 (8K) has a token limit of 8000, and GPT4 (32K) has a token limit of 32000. Exceeding the token limit would require splitting the text into smaller chunks and processing them separately, which may affect the quality and coherence of the output.

What is Prompt?

A prompt includes the instructions or questions you give the model, and it can also have extra details like context, examples, or inputs. Using these elements effectively helps you get better results.

Basics of the prompting

You can achieve a lot with simple prompts but a lot more can be achieved if we tune the prompt with more context, inputs, or examples. Here are few examples of prompts:

Prompt
What is the color of grass?
Output
Green

Prompt
This is awesome! // Positive
This is bad! // Negative
Wow that movie was rad! // Positive
What a horrible show! //
Output
Negative

The above example is the example of few shots prompting, where we pass few examples in the prompt to get the better results. Here we can see that last sentence is correctly classified as "Negative".
Note: Output can vary if different LLMs are used.
In this article, we can see how to use various prompt engineering techniques to improve LLMs SQL generating capabilites. Now getting an SQL query from natural language query can be broken down into three major problems.

- **Entity Extraction**: Retreiving the correct Entity to be used in SQL Query.
- **SQL Generation**: Generating the logically correct SQL.
- **Validation Methods**: Validation the SQL and generated output.

Entity Extraction

In this segment, we will explore the process of obtaining the accurate entities for use in SQL. Imagine you have a database containing several tables, also referred to as entities, in the e-commerce industry.

Few Entities Case

In this sample case where we don't have many entities; we can pass the all the entities in SQL generation prompt. **Voorbeeld 1**
When the schema and number of tables is small, there is no real need to do any entity extraction as the entire schema can be put into the prompt. Few shots prompting for entity extraction.

```
SYSTEM
Given the following SQL tables, your job is to write queries given a user's request.

CREATE TABLE Orders (OrderID int, CustomerID int,OrderDate datetime,OrderTime
varchar(8),PRIMARY KEY (OrderID));
CREATE TABLE OrderDetails (OrderDetailID int, OrderID int,ProductID int,Quantity
int,PRIMARY KEY (OrderDetailID));
CREATE TABLE Products (ProductID int, ProductName varchar(50),Category
varchar(50),UnitPrice decimal(10, 2),Stock int,PRIMARY KEY (ProductID));
CREATE TABLE Customers (CustomerID int, FirstName varchar(50),LastName
varchar(50),Email varchar(100),Phone varchar(20),PRIMARY KEY (CustomerID));

USER
Write a SQL query which computes the average total order value for all orders.
```

1

```
Prompt

Act as a helpful assistant who specializes in SQL entity extraction.
Here are all the tables.
Users (user_id, username, email, password, registration_date)
Products (product_id, product_name, price, description, category_id)
Categories (category_id, category_name)
Orders (order_id, user_id, order_date, total_amount)
OrderDetails (order_detail_id, order_id, product_id, quantity, subtotal)
Addresses (address_id, user_id, street_address, city, state, postal_code)
Reviews (review_id, product_id, user_id, rating, review_text, review_date)
Employees (employee_id, first_name, last_name, hire_date, job_title, manager_id)
Departments (department_id, department_name, location)
Projects (project_id, project_name, start_date, end_date, department_id)
Customers (customer_id, first_name, last_name, email, phone)
Suppliers (supplier_id, company_name, contact_name, contact_email, phone)
Invoices (invoice_id, customer_id, invoice_date, total_amount)
InvoiceDetails (invoice_detail_id, invoice_id, product_id, quantity, unit_price,
total)
Transactions (transaction_id, account_id, transaction_date, transaction_type,
amount)
Accounts (account_id, account_number, account_type, balance)
Books (book_id, title, author, publication_date, genre)
Students (student_id, first_name, last_name, birth_date, major)
Teachers (teacher_id, first_name, last_name, hire_date, department)
Events (event_id, event_name, event_date, location, organizer)

Note: Do not use any table not listed above.

Query: Find the names of all users who have placed an order in the last month.
Tables: Users Table, Orders Table

Query: List the products in the "Electronics" category with a price less than $500.
Tables: Products Table, Categories Table

Query: Show the total sales for each category in the past year.
Tables: Products Table, Categories Table, Orders Table, OrderDetails Table

Query: Retrieve the contact information of all customers who have overdue invoices.
Tables: Customers Table, Invoices Table, InvoiceDetails Table

Query: Find the titles of books written by the author "J.K. Rowling
Tables :

Give me the tables for the last query. Only give the list of table name in the
output , nothing else.
```

2

We use this method when we have enough entities which we can't fit in prompt. So, we extract only the required one, so it can be sent to SQL generation prompt. Here in the prompt, we will have "K" examples which will have the natural language ask and corresponding entities used to generate the SQL. **Voorbeeld 2**
The above are just a couple of simple prompting techniques. There are more advanced options available which might be more useful in an existing production scenario.

SQL Generation

SQL generation can be a quite tricky task. We can also use a few shots prompting here as well. Just like with entity extraction, if you only have a couple of examples all of them can be put into the prompt. However, if you have a lot of examples available, this will quickly use up too many tokens. When this is the case, you need to do some kind of similarity search. The preferred way of doing similarity search is by creating embedding vectors for examples, and searching for similar entries based on user's request. Implementing such a feature is a topic by itself and too advanced to put into this article. Therefore, I'll provide a hard-coded list of examples in the scenario over here. Similarity search to fetch top k most similar examples. **Voorbeeld 3**

Result validation methods.

I've seen this prompt work for syntax errors, but it will not solve every error imaginable for a query. It's good to know the query can still fail when executing, so you should handle this like always.

BIO

Dhruv Patel

I'm Dhruv Patel, a recent graduate and a fresher at Microsoft India. I'm on a learning journey to master Azure, Large Language Models (LLMs), and explore Azure Databases. Join me as I share my experiences and insights on these cutting-edge technologies and my path to becoming a tech expert.
You can find more of me on LinkedIn at https://www.linkedin.com/in/dhruv-patel-110002191/

Prompt

3

```
Act as an expert SQL database engineer.
Your task is to create valid, well-formed SQL queries based on given examples and
a provided database schema. Use explicit table aliases where possible and avoid
duplicating column names across joins; make them descriptive and unique. Reference
only the provided database schema and avoid referencing anything else that wasn't
defined in the schema, as it may cause errors.

Here are few examples, use them to understand the syntax and SQL generation method.
```
Question : Find the names of all users who have placed an order in the last month.
SQL : ```SELECT DISTINCT u.username
FROM Users u
INNER JOIN Orders o ON u.user_id = o.user_id
WHERE o.order_date >= DATEADD(MONTH, -1, GETDATE());```

Question : List the products in the "Electronics" category with a price less than
$500.
SQL : ``` SELECT p.product_name
FROM Products p
INNER JOIN Categories c ON p.category_id = c.category_id
WHERE c.category_name = 'Electronics' AND p.price < 500;```

```

You will be provided with the SQL used to create the schema of an existing database.

SQL used to create Database Schema:
```
{{$schema}}
```
Question: {{$input}}

The SQL should be valid, well-formed, and execute without error.
Your response will be executed as is so only output SQL that is immediately
runnable.
Analyze from the perspective of other experts in your field and work it out in a
step by step way to be sure you have the right answer.
Do not format or add commentary to your response. No prose.
```

Prompt

4

```
Act as an expert SQL database engineer.

You will be provided with the SQL used to create the schema of an existing database.
You will also be provided with the SQL that was attempted to be run against the
database, but resulted in an error.
Your goal is to fix the SQL so that it is valid, well-formed, and executes without
error.

Schema:
```
{{$schema}}
```

Query:
```
{{$input}}
```
Error: {{$error}}

Ensure the SQL is valid, well-formed, and executes without error.
Analyze from the perspective of other experts in your field and work it out in a
step by step way to be sure you have the right answer.
If there is nothing to fix, output the query as is without any modifications.
Do not format or add commentary to your response. No prose.
```

Prompt

5

```
Act as a helpful assistant who specializes in finding information within tabular
data and communicating that information to an end user in layman's terms.

You will be provided with tabular data and the SQL that was used to query it from an
existing database.
Your goal is to answer the question using only the provided data.

SQL:
```
{{$sql}}
```

Data:
```
{{$data}}
```
Question: {{$input}}

Only produce ouput that directly answers my question.
If the data doesn't answer the question, or you don't know the answer, tell me.
Don't try to make up an answer.
```

Fix generated SQL.

At times, language models may produce incorrect queries. Language models can be employed to rectify SQL queries by taking the error message returned from a database call and using it as input to the model. **Voorbeeld 4**

Information extraction from the generated SQL table data.

Now, a SQL would return the tabular output, which might now be relevant for many normal users , so this prompt can produce natural language response to the user query using the data generated by the SQL query.app **Voorbeeld 5**

Conclusion

I've observed substantial benefits in employing LLMs for crafting queries based on natural language requests. This opens numerous possibilities for both users and engineers. However, it's important to remain mindful of concerns related to security, governance, injection, accuracy, and correctness when integrating LLMs into your or your company's solutions.
If you're contemplating how to integrate LLMs into your business, the NL-to-SQL approach detailed in this article serves as an excellent starting point, but it's not the destination. In my own work, I've utilized widely available models such as GPT-3.5 and GPT-4, My recommendation is not to fixate too much on a single solution or model, but instead, explore and ascertain what works best for your particular needs. ●

References

https://github.com/anthonypuppo/sk-nl2ef-plugin
https://www.promptingguide.ai/

Artificial intelligence **and dial phones**

In mid-October, I found myself in Cluj-Napoca, Romania, speaking at the Voxxed Days conference. It wasn't all just work. I had the chance to belt out tunes at a karaoke bar named Flying Circus (a nod to Monty Python). Besides that and my closing keynote, the real highlight for me, however, was participating in an on-stage forum discussion on the use of artificial intelligence in software development.

Auteur: Sander Hoogendoorn

The opinions on AI tools among the forum participants and the audience alike were all over the map. Some worried about intellectual property, while others believed in limiting access to tools such as ChatGPT and GitHub Copilot, citing potential dangers.
The most intriguing question came from a member of the audience. He wondered if using AI tools might prevent young developers from acquiring essential coding skills, as these tools automate the basics away from them.

As a daily user of Copilot and ChatGPT, I can attest to their time-saving capabilities. They excel in suggesting solutions, like when I'm writing unit tests. Even though these suggestions sometimes make no sense at all, more frequently my AI assistants take away lots of typing effort. And that's a plus.
But, I do disagree with the member of the audience that AI will automate things we need to teach our youngsters. Yes, AI will eliminate the mundane tasks that both junior and senior developers now perform manually.

BIO

Sander Hoogendoorn

```
{
 "name": "Sander Hoogendoorn",
 "motto": "Small steps are the fastest way forward",
 "roles": ["Software Craftsman", "CTO", "Speaker", "Writer"],
 "phone": "+31637740255",
 "web", ["sanderhoogendoorn.com", "flowmanifest.org"],
 "linkedin": "linkedin.com/in/aahoogendoorn/",
 "twitter": "twitter.com/aahoogendoorn",
 "github": ["github.com/aahoogendoorn","github.com/thisisagile"],
 "talks": "sanderhoogendoorn.com/my-current-talks/"
}
```

But is that such a bad thing? With AI reshaping our industry, should junior developers still focus on learning these vanishing skills?
This reminds me of a recent conversation with my football buddies. One of them lamented that today's youth can't handle basic calculus. I countered that they've found alternative ways to solve problems without it, and they seem to be doing just fine.
Later that evening in Cluj we had dinner in a nice restaurant. As decoration, on the wall in the restaurant was a classic rotary dial phone. A nice example of skills my older generation still has in their muscle memory, but younger generations have no idea about it. As an experiment the conference organizer invited one of the younger speakers, in his late teens, to use the device to call someone up. It was an interesting experiment to watch. The youngster seriously struggled to use this strange device. After having tried pushing the numbers on rotary the dial a couple of times, he gave up. It never entered his mind to turn the dial. Let alone pick up the phone first.
Yet he was effortlessly navigating a state-of-the-art iPhone 15. The contrast was clear.
So, does AI change the way we work? Absolutely. I noticed it on my flight to Cluj when I was writing code and I'd unconsciously pause, expecting Copilot to complete my code, only to realize there was no Wi-Fi. It's already ingrained in my muscle memory.
And will AI prevent people from learning the current basics of coding? Yes, it will, and that's not a big problem. We won't return to these basics. In similar ways, people stopped learning how to ride horseback when cars were first introduced. In a few years or maybe even sooner, AI will offer a higher level of abstraction to developers, saving us time to do what we developers excel at – solving problems, not pressing buttons on dial phones. ●

Lessen uit de praktijk: hoe integreren we Azure API Management met onze CI/CD-processen?

Sinds een aantal jaren is Azure API Management haast onvermijdelijk voor iedereen die API's ontwikkelt en deployed in Azure. Het is dan ook een fantastische one-stop-shop voor allerlei aspecten die samenhangen met het beheer, de publicatie en de consumptie van API's, die je écht liever niet in code regelt. Azure API Management functioneert als Gateway, waarbinnen je bijvoorbeeld caching, throttling of authenticatie kunt regelen. Maar denk ook aan aggregaties, dus het combineren van meerdere backend API's in één uniforme publieke API, bijvoorbeeld als Backend for Frontend. Of aan het vermarkten van je API's door middel van subscriptions op (subsets) van API's die je ter beschikking stelt.

Auteur: Annejan Barelds

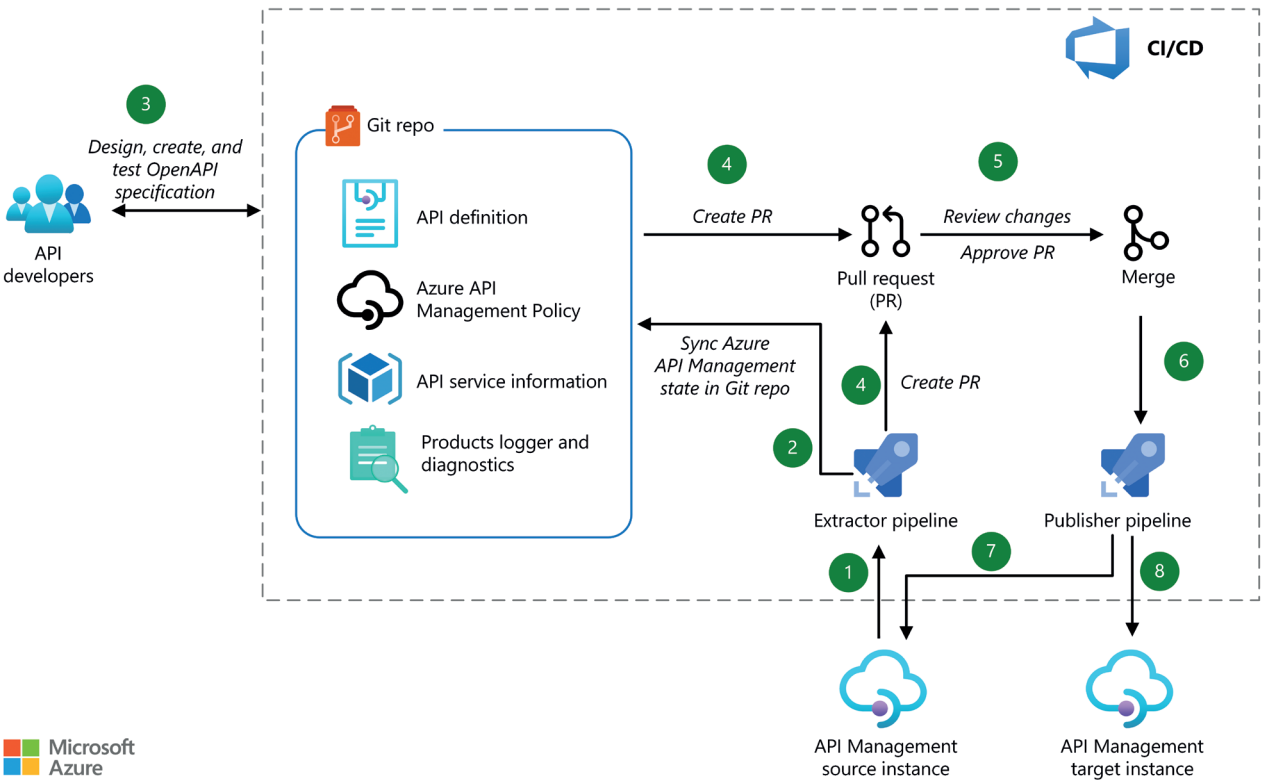
Maar API Management is óók een tamelijk complex product, en het publiceren van een API door middel van API Management betekent dat je al snel meerdere sub-resources in API Management moet registreren. Uiteraard de API definitie zelf, maar ook zogenaamde Backend resources (om backendsystemen te ontsluiten), Policies (om het runtime gedrag te beïnvloeden), Named Values, Products, enzovoort.
Dus iedereen die ervoor kiest om Azure API Management in te zetten voor de invulling van een of meer van bovenstaande requirements, moet daarna bedenken hoe de spulletjes in Azure API Management tot stand komen (wie ontwikkelt het een en ander, en hoe?), en hoe dat vervolgens gecontroleerd uitgerold wordt om uiteindelijk in productie genomen te worden.

De afgelopen maanden ben ik bezig geweest om dit probleem op te lossen voor het team waar ik mee werkte. En ik denk dat de gekozen oplossingen, de bijbehorende afwegingen en lessons learned, relevant kunnen zijn voor een breder publiek. Daarom bespreek ik in dit artikel hoe we CI/CD en DevOps principes kunnen toepassen op Azure API Management (APIM).
In dit artikel kom je dus géén informatie tegen over waarom je Azure API Management zou moeten gebruiken. Het uitgangspunt is dat die redenen er zijn, en dat je daarom een oplossing moet vinden voor de gecontroleerde uitrol ervan.
Er hoort een flinke hoeveelheid code bij dit artikel, die we niet allemaal kunnen tonen. Maar alles is te vinden op GitHub: https://github.com/AnnejanBarelds/ManagingAPIM.

APIOps

In ons team stelden we een aantal eisen aan het proces rond de ontwikkeling en deployment van API's in APIM. We wilden graag zoveel mogelijk de Azure portal (op de development omgeving) kunnen gebruiken om API's, en alles wat daarbij hoort, in te regelen. Deze configuratie moet vanuit de development omgeving geautomatiseerd over de andere omgevingen uitgerold kunnen worden. Bij voorkeur door middel van Azure Pipelines, want dat gebruiken we ook voor alle overige CI/CD-processen. Daarbij zou het wenselijk zijn als we een aantal validatiestappen in konden bouwen om de meest voorkomende fouten tijdens de deployment op te sporen. En, we wilden een oplossing die naadloos integreert met de gekozen versioneringsstrategie op de backend.

Voor de geautomatiseerde uitrol kwamen we al snel uit bij APIOps (https://github.com/Azure/apiops). Dit is een toolset die DevOps-concepten toepast op APIM, en eigenlijk in de basis precies doet wat we willen: er is een bronomgeving (de APIM-instantie op development) van waaruit we exports doen van alle subresources – dus de API's die we in APIM hebben gedefiniëerd, maar ook onder andere de Backend resources, Policy Fragments, Named Values, VersionSets, enzovoort. Deze export brengen we onder in Git, van waaruit we kunnen importeren in één of meer doelomgevingen. Een plaatje maakt goed duidelijk hoe dat proces werkt.



De processtappen zijn als volgt.

> Een Extractor pipeline exporteert de subresources uit de APIM-instantie die we als bron gebruiken – in ons geval de development omgeving.
> De pipeline maakt een feature branch, waarin de stand van zaken in de export wordt vastgelegd.
> Eventueel kunnen developers zelf ook rechtstreeks vanuit code aanpassingen doen.
> Zowel code-changes door developers als exports vanuit de pipeline, worden via een PR aangeboden.
> De PR wordt bekeken en goedgekeurd.
> De PR wordt gemerged naar main, en triggert daarmee een Publisher pipeline.
> De laatste stand van zaken wordt eerst weer uitgerold naar de bron-omgeving – met name om ervoor te zorgen dat ook eventuele handmatige changes uit stap (3) ook op development terecht komen.
> De changes worden uitgerold over één of meerdere andere doel-APIM-instanties.

APIOps biedt Azure DevOps pipeline templates aan, en ook workflows voor GitHub Actions. Omdat wij Azure Pipelines al gebruiken, kiezen wij hiervoor. De rest van het artikel zal dan ook daarop gebaseerd zijn, maar ongetwijfeld zijn vergelijkbare dingen te doen met de GitHub Actions workflows.

Bicep vs APIOps

Met het gebruik van APIOps voor de uitrol van APIM subresources ontstaat wel een tweedeling tussen resources in Azure die we met traditionele Infrastructure-as-Code-oplossingen uitrollen, en resources die we middels APIOps uitrollen. Uiteraard rollen we de APIM instanties zélf uit middels IaC (in ons geval Bicep). We hebben, met andere woorden, een shared resources pipeline die de APIM instanties op zowel development, als acceptance en production uitrolt. Aan de andere kant van het spectrum hebben we de API subresources die we in APIM definiëren; die worden uiteraard met APIOps uitgerold. Maar hoe gaan we om met – bijvoorbeeld – Backend subresources?

We hebben voor de daadwerkelijke backends (bijvoorbeeld Azure Web

Apps) service pipelines die de Azure Web App uitrollen, en in één moeite door een Backend subresource definiëren in APIM. Waar leggen we de verantwoordelijkheid voor het uitrollen van deze Backend-resources in acceptance en production? Is dat de verantwoordelijkheid van de pipeline die ook de Azure Web App uitrolt? Maar dan moeten we deze backend subresource niet óók nog eens gaan exporteren en importeren middels APIOps; we willen niet eindigen met twee processen die dezelfde verantwoordelijkheid hebben.

APIOps biedt de mogelijkheid om voor subresources een config file bij te houden, waarin je op basis van resource type en name vastlegt welke resources geëxporteerd moeten worden; alle subresources die niet in die configuratie voorkomen, worden overgeslagen. Dat zou het probleem van de dubbele verantwoordelijkheid oplossen, maar het leidt wel tot een wirwar aan administratie in de configuratie – en developers moeten dan onthouden welke subresources door APIOps worden geëxporteerd, en welke niet. Tot slot zou het ook betekenen dat we de Backend resources niet in Git zouden hebben staan, en dat betekent dat we op basis daarvan ook geen deployment-beslissingen kunnen nemen. En zoals we nog zullen zien: dat willen we wel graag.

Er kleven dus nogal wat nadelen aan de optie om bepaalde subresources wel of juist niet mee te nemen in de export. We kiezen er daarom voor om per definitie alle subresources mee te nemen in APIOps, en als gevolg van die keuze configureren we de service pipelines zó dat alleen op development de Backend subresource in APIM wordt uitgerold; op alle andere omgevingen wordt deze stap overgeslagen. Zo komen we weer op een consistente verdeling van verantwoordelijkheden. **Voorbeeld 1**

Overigens gelden vergelijkbare overwegingen ook voor andere subresources, zoals Named Values: altijd geldt dat deze door de IaC pipelines alleen in development uitgerold worden; van daaruit worden ze verder opgepakt door de APIOps pipelines.

Breaking changes detecteren

Nu we een sluitende strategie hebben om al onze spulletjes op een consistente manier door de verschillende omgevingen heen te loodsen, gaan we werken aan een aantal checks-and-balances in de APIOps pipelines. Out-of-the-box is al een **Spectral** linter (https://stoplight.io/open-source/spectral) opgenomen. Deze linter checkt de OpenAPI specification files (die onderdeel zijn van de export) op issues, en wordt uitgevoerd in de **Extractor pipeline**. Maar natuurlijk zijn er meer scenarios te verzinnen waarbij we willen ingrijpen in het export-import proces. Het meest voor de hand liggend zijn breaking changes in de API definities. We hebben straks developers die in de APIM portal op de development omgeving API's gaan maken en aanpassen. En uiteraard doen zij allemaal hun best om bestaande clients niet te breken als gevolg van een API-aanpassing, maar een foutje is snel gemaakt. Daarom richten we een proces in om die OpenAPI specification files niet alleen te linten, maar ook te vergelijken met de versies in de main branch. Hiervoor gebruiken we **openapi-diff** (https://www.npmjs.com/package/openapi-diff). Wij hebben ervoor gekozen om dit uit te voeren als onderdeel van een PR build, en door middel van een PR

BIO

Annejan Barelds

Annejan werkt als developer, architect en cloud consultant bij 4Dotnet, en in die rol helpt hij klanten om goede software te ontwikkelen en optimaal gebruik te maken van de cloud. Hij werkt nu ruim 15 jaar al s developer. Zo'n 10 jaar geleden kwam hij in aanraking met Azure, en het enthousiasme over die grote bouwblokkendoos is sindsdien niet meer weg geweest. Hij houdt van het maken van technische oplossingen, maar ook van het delen van zijn kennis en ervaring met anderen.

```
module apimBackend 'ApiManagement/service/backend.bicep' = if(environment == 'dev')
{
  name: '${deployment().name}-apimBackend'
  scope: az.resourceGroup(apimResourceGroupName)
  params: {
    name: replace(webapp.outputs.name, '-${environment}', '')
    apimName: apimName
    webAppName: webapp.outputs.name
    webAppResourceGroupName: resourceGroup.name
  }
}
```

1

> Vind deze en de andere listingen uit het artikel op sdn.nl!

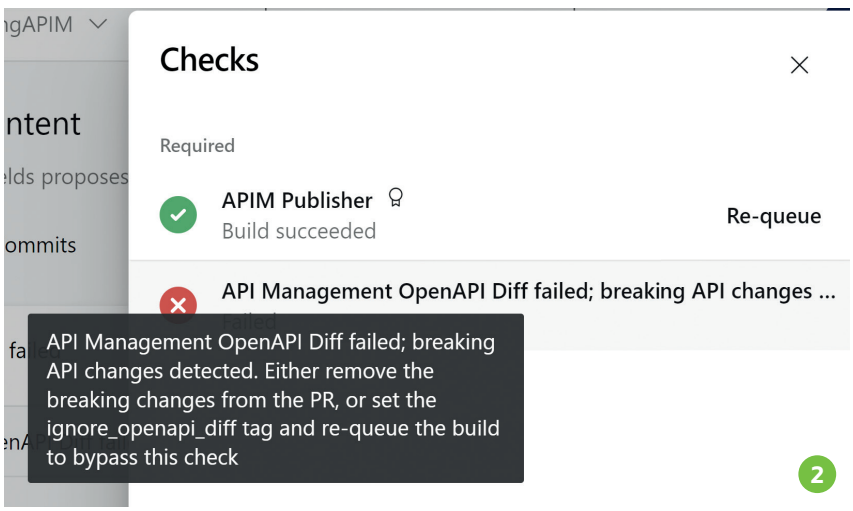
Status het resultaat te communiceren. **Voorbeeld 2**
Hierdoor kan de PR niet gemerged worden met main als er breaking changes gevonden worden tussen een OpenAPI specification file in de PR enerzijds, en zijn equivalent in main anderzijds. Uiteraard kan dit gedrag uitgeschakeld worden. Wij hebben ervoor gekozen om dit middels een PR Tag te doen: als de PR een bepaalde tag heeft, wordt de OpenAPI diff nog wel uitgevoerd, maar de PR Status wordt genegeerd en dus kan de PR gemerged worden. Op deze manier is het nog steeds mogelijk om breaking changes door te voeren, maar de developer en de PR-reviewer moeten daar nu in elk geval heel bewust voor kiezen.

Verouderde backend-versie

En er kunnen natuurlijk meer zaken fout gaan. Ik legde al uit dat de API's in APIM bij ons doorverwijzen naar Azure Web Apps, en voor elke Web App die we ontsluiten, registreren we een Backend resource in APIM. Maar wat gebeurt er als we een Web App deployen naar development en acceptance, in APIM een API definiëren op basis hiervan, deze exporteren, en vervolgens importeren in acceptance én production? Juist, de API zal breken op production, want daar staat de Web App nog helemaal niet.

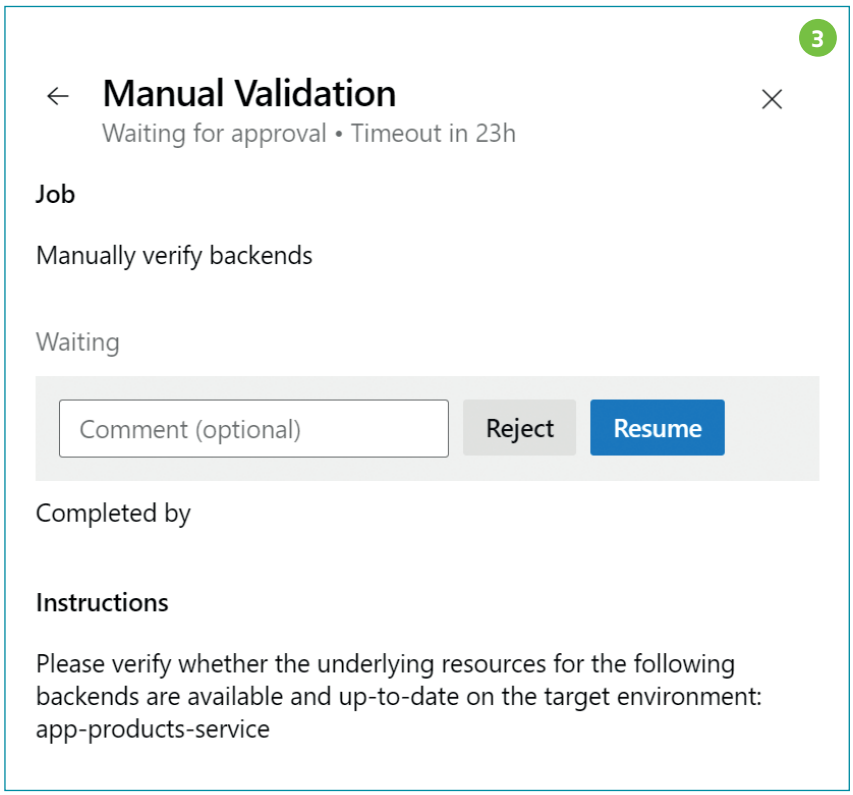
En uiteraard geldt hetzelfde wanneer de Web App al lang live staat, maar een nieuwe feature nog niet volledig uitgerold is.
Om ongelukken van dit type te voorkomen, hebben we extra stappen ingebouwd op verschillende momenten in de APIOps pipelines.

> In de **Extractor pipeline** hebben we ingebouwd dat, voor elke geëxporteerde backend resource, de onderliggende Azure Web App resource wordt opgezocht. Van die resource leggen we de waarde van de Version tag vast. (Deze waarde moeten we dus wel ook zetten bij de deployment van de Azure Web App in de betreffende service pipeline.) Deze versies slaan we op in een YAML configuration file voor gebruik in de volgende stap.
> In de **Publisher pipeline** hebben we, voor elke omgeving anders dan development, extra jobs ingebouwd. In de eerste job valideren we dat, voor elke APIM backend resource
  - de achterliggende Azure Web App bestaat;
  - en dat deze minimaal dezelfde versie heeft als de versie die we in stap (1) hebben opgeslagen.
> Als aan één van deze twee condities niet wordt voldaan voor minimaal één backend, bereiden we een gebruikerswaarschuwing voor.
> In de tweede job checken we of er een gebruikerswaarschuwing in de pipeline context aanwezig is. Zo niet, dan gaan we moeiteloos door naar de laatste job – de daadwerkelijke import in de betreffende APIM instantie. Maar als er wél een waarschuwing is, starten we een Manual Validation task met daarin de waarschuwing. Om de import alsnog te starten, moet nu iemand een actieve handeling verrichten. Dus ook hier geldt: een developer (of wie dan ook verantwoordelijk is voor de uitrol) kan prima een import doen terwijl bepaalde achterliggende systemen niet aanwezig of niet up-to-date zijn, maar daar moet dan wel bewust voor gekozen worden. En meestal zal de juiste reactie zijn: 'ah, ja, eerst die service pipeline nog even doorzetten'. **Voorbeeld 3**



Enkele eigenaardigheden

Tot zover klinkt het allemaal heel mooi en robuust. En dat is het ook – als je rekening houdt met een paar eigenaardigheden.
Zo werkt de **Publisher pipeline** in APIOps in principe altijd op de laatste commit op de repository. En dat heeft een reden: het is de enige manier waarop vastgesteld kan worden welke resources eventueel verwijderd moeten worden uit APIM. Als het importproces altijd simpelweg naar de huidige stand van zaken in main zou kijken, dan zouden alle resources die daarin staan, bijgewerkt worden; resources die daar niet in staan, zouden ongemoeid gelaten worden. (Uiteraard zou het ook een optie zijn om alle resources die niet in main staan actief te verwijderen, maar dat breekt dan weer de eerder besproken APIOps-functionaliteit om sommige subresources niet mee te nemen in een export. Dus bottom line is dat APIOps wel móét werken op basis van de laatste commit, om ook deletions mee te krijgen.)
Maar dat heeft wel een paar gevolgen. Zo betekent dat dus dat APIOps



ervan uit gaat dat je een aparte repo hebt met daarin je APIM resources. Want bij een repo waar ook andere spullen in staan, kun je in de situatie komen dat een APIOps commit direct gevolgd wordt door een code commit op iets heel anders. Als daarna de publisher gaat lopen, kijkt-ie naar de verkeerde commit. Een manier om daar omheen te werken, is het inbouwen van een taak die de laatste commit op het specifieke pad naar de APIM resources opzoekt. Daar hebben wij voor gekozen, maar je kunt er dus ook gewoon voor kiezen om de APIM resources in een eigen repository onder te bengen.
Een ander gevolg van het importeren op basis van de laatste commit, is dat de publisher breekt als het verwijderen van de resource in de APIM-bronomgeving is gedaan. Het proces verloopt dan namelijk als volgt:

> Een developer verwijdert een resource (bijvoorbeeld een API) uit APIM.
> De **Extractor pipeline** pikt deze delete op, en via een PR komt-ie in main terecht.
> De **Publisher pipeline** reconstrueert op basis van de laatste commit dat de betreffende API verwijderd moet worden.
> Tijdens de import naar development wordt een HTTP DELETE uitgevoerd op een resource die in stap (1) al verwijderd was – en die faalt dus.

Met andere woorden: je kunt als developer bijna alles doen in de APIM-bronomgeving, maar deletions moet je rechtstreeks in Git doen. Want dan komt-ie net zo goed in main, wordt-ie net zo goed opgepakt door de importer, maar wordt de DELETE uitgevoerd op een resource die nog bestaat. Gelukkig is de folderstructuur in Git heel overzichtelijk; elk resource type zit in een eigen folder, en daarbinnen zit elke resource ook weer in een eigen folder. Het verwijderen van een resource is dus niet meer dan het verwijderen van een folder in Git.
Nog een gevolg is dat APIOps pipelines, anders dan we gewend zijn van de meeste code deployment pipelines, cumulatief is. Dus als je verschillende exports en imports doet voordat je naar production gaat, moet je alle imports sequentieel draaien.

Versionering

Een laatste aandachtspunt is de versionering van API's. Aan de achterkant, dus in de code die draait op Azure Web Apps, maken we gebruik van ASP.NET API Versioning (https://github.com/dotnet/aspnet-api-versioning) op basis van een api-version query string. Die optie wordt ook ondersteund wanneer je versioning in APIM gaat toepassen. Andere opties (zoals op basis van een versie-aanduiding in het pad) zijn uiteraard ook mogelijk in zowel APIM als ASP.NET API Versioning, maar die leiden tot allerlei URL rewriting policies, dus daar wilden we bij voorkeur weg van blijven.
En dat kan. Je moet je dan alleen wel houden aan een extra 'spelregel' in

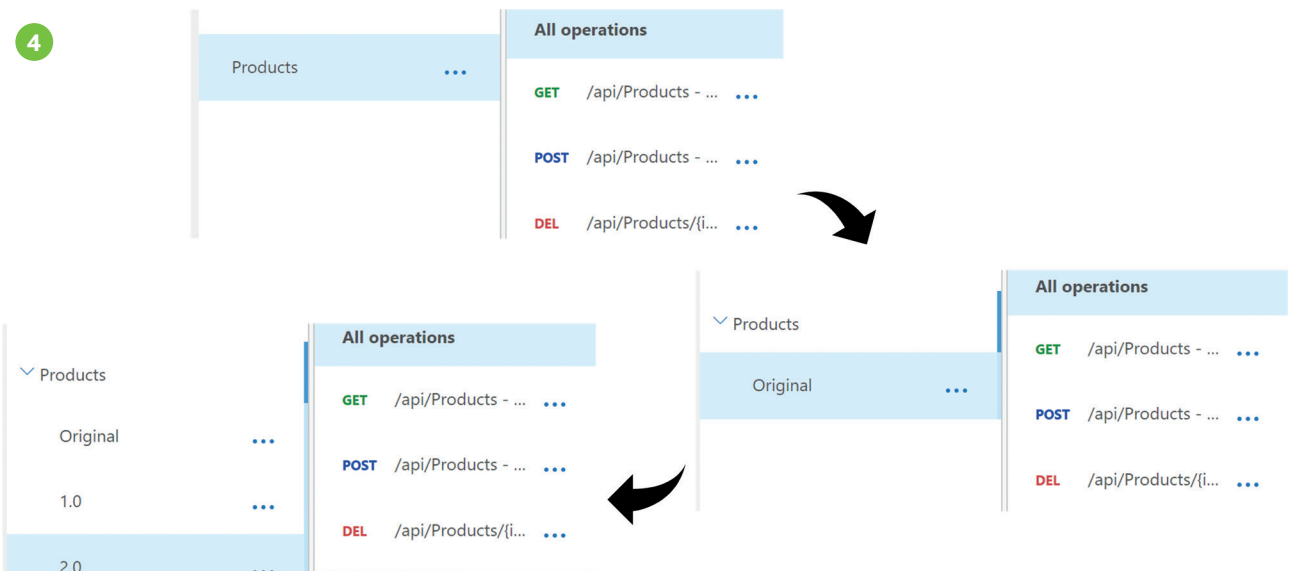
Azure API Management is onvermijdelijk voor iedereen die API's ontwikkelt en deployed in Azure

het gebruik van APIOps. Het voert te ver om hier de precieze interne werking van APIM rond versionering af te pellen, en met die kennis te bespreken hoe dat invloed heeft op APIOps. Dus we laten het hier bij een algemeen advies.
In mijn ervaring beginnen teams vaak met een ongeversioneerde API. Versies komen pas aan de orde wanneer er een breaking change doorgevoerd moet worden, en het geen optie is dat alle clients in een 'big bang' overgaan. Dat heeft als gevolg dat je dus ook in APIM begint met ongeversioneerde API's, en pas ergens gedurende de API lifecycle over wilt naar een geversioneerde variant. Als je kiest voor een versioneringsstrategie op basis van een query string, én je maakt gebruik van APIOps, dan moet je dat gefaseerd aanpakken. Dat wil zeggen, je wilt éérst van een ongeversioneerde API overstappen naar een geversioneerde API met daarin alleen de originele API. Pas nadat je die uitgerold hebt over al je omgevingen, kun je extra versies toevoegen.
Voor wie écht wil weten waarom dit zo is, toch even een heel korte verklaring. Als je direct meerdere versies toevoegt, wil de APIOps **Publisher pipeline** meerdere API-versies uitrollen – maar deze delen allemaal dezelfde URL. En dat wil APIM niet, tenzij de API's onderdeel zijn van dezelfde VersionSet. En hoewel die VersionSet ook onderdeel is van dezelfde import, is die informatie binnen APIM op dat moment nog niet bekend. Dus uiteindelijk gaat dit om een volgordelijksheidsprobleem in de APIOps importer.
Overigens: ook hier gelden dezelfde regels over deletions. Dus de eerste stap van een ongeversioneerde API naar een geversioneerde API moet écht uitmonden in een geversioneerde API met daarin alleen de originele API, en dus niet met daarin alleen v1, of v2. Want under the hood zou dat een DELETE op de originele API zijn – en daar breekt APIOps op bij het importeren naar de bronomgeving.
Visueel ziet dat stappenplan er dus als volgt uit: **Voorbeeld 4**
Uiteraard kun je dit ongemak voorkomen door van het begin af aan te werken met geversioneerde API's, zelfs als je nog maar één versie hebt.

Tot slot

In dit artikel hebben we bekeken hoe je, door middel van APIOps, de bekende CI/CD en DevOps principes kunt toepassen op Azure API Management. APIOps biedt hiervoor een prachtige basis, en is uitbreidbaar met je eigen checks-and-balances die je ook gewend bent in te bouwen in je andere pipelines. Het komt met een aantal eigenaardigheden en do's en don'ts, maar als je je daarvan bewust bent is APIOps een krachtige tool om gecontroleerd je Azure API Management-inrichting door de verschillende stadia in productie te brengen.
De volledige verzameling Bicep templates, YAML templates, inclusief een voorbeeld-API, is beschikbaar op GitHub, te vinden onder de volgende QR code: ●

1. Afbeelding gebaseerd op https://learn.microsoft.com/en-us/azure/architecture/example-scenario/devops/automated-api-deployments-apiops



Interview met Arjen de Ruiter over zijn Technical debt spel

Elke developer heeft wel te maken met technical debt. Iets wat niet altijd gemakkelijk op de agenda komt bij het management. Arjen de Ruiter stelt daarentegen dat technical debt niet altijd als iets slechts gezien hoeft te worden. Een interessant tegengeluid van iemand die zelf developer is geweest. Daarnaast heeft hij een spel ontwikkeld over technical debt. Genoeg redenen voor mij om hem hierover te interviewen.

Arjen, laat ik direct met de deur in huis vallen. Waarom is technical debt niet altijd iets slechts?
Er zijn een paar redenen waarom technical debt niet per definitie slecht is. Eén reden is dat je soms snel iets live wilt brengen om te toetsen of gebruikers het een goede oplossing vinden. Als je nog niet weet of het een succes wordt, dan kan je misschien wat shortcuts nemen op non-functionals zoals schaalbaarheid. Dan bouw je dus wat technical debt op. Als je product dan geen succes is, dan heb je dat redelijk goedkoop getoetst en dan kan je eerst daaraan werken. Als je product wel een succes is, dan heb je een luxeprobleem en moet je werken aan die technical debt. Dus omwille van je time to market is technical debt niet slecht.
Een andere reden is dat een bepaalde technische keuze vroeger de juiste was, maar met de kennis van nu niet meer. Dan is de technical debt langzaam ontstaan i.p.v. een bewuste keuze. Essent had bijvoorbeeld oplossingen die vroeger zogenaamd low-interest waren. Mensen keken zelden naar hun verbruik. Kosten waren voorspelbaar. Nu de energieprijzen meer bewegen en energietransitie voor iedereen een belangrijk onderwerp is, zijn de oplossing van Essent

high-interest geworden. Daarmee is traffic enorm toegenomen en bleek onze oplossing niet zo schaalbaar. Maar ja, voor de situatie destijds was het een prima oplossing. Inmiddels is het gelukkig opgelost.

Arjen de Ruiter heeft ruim 5 jaar bij Bol.com gewerkt (2011 – 2016) als Software Development Manager. Daarvoor heeft hij 13 jaar gewerkt als Developer. Ook na zijn rol bij Bol.com heeft hij niet stil gezeten en gewerkt bij Sendcloud als VP Engineering, bij Maykers (onderdeel van Kramp) als Head of Product & Engineering en bij Coosto als CTO. Momenteel werkt Arjen als Head of Software Delivery bij Essent.

Zou je meer kunnen vertellen over het technical debt spel dat je ontwikkeld hebt?

Het spel gaat over snel (software leveren) versus goed (kwaliteit van de software): het gaat erom balans te vinden tussen deze twee. Daarnaast zit er nog een geluks-/gunfactor in aangezien in het echte leven ook niet altijd alles volgens plan loopt.

Wat was voor jou de aanleiding om het Technical debt spel te maken?

Het spel is ontwikkeld tijdens de periode dat ik bij Bol.com werkte. Ik werkte toen met Agile coaches. Het viel me op dat er veel discussies gaande waren over keuzes gemaakt in het verleden en ik had het gevoel dat we vast bleven hangen in die discussies. Er werd veel geklaagd over de grote hoeveelheid technical dept terwijl juist die "oude" systemen het grootste gedeelte van de omzet binnen bracht. Ondertussen ging de velocity van de teams bijna volledig op aan het in de lucht houden houden van de huidige verouderde systemen.
Het is de bedoeling om teamleden met de stakeholder samen het spel te laten spelen. Dit kan je een keertje doen ipv een retro. Het gaat vooral om het gesprek dat je hebt na het spelen van het spel.
Door het spel te spelen, krijgen de spelers het inzicht dat het om de balans gaat.
Als je voor 1 van de beide kiest dan heeft dat zijn nadelen. Als je voor snel gaat, kom je in de toekomst in de problemen omdat het steeds lastiger wordt om features op te leveren en je systemen trager worden. Als je voor goed gaat, loop je kans dat je aan het "gold plating" bent waardoor je de boot mist.

Waarom heb je er een spel van gemaakt en niet in de vorm van een talk of artikel?

Talks over technical debt heb ik ook gehouden. Echter talks zijn erg passief terwijl je wilt dat mensen ermee aan de slag gaan en discussies hierover gaan voeren. Het spel is juist interactief en zorgt voor veel meer diepgang. Door het spel te spelen ervaren de spelers het (de balans opzoeken tussen snel versus goed) zelf. Het gaat vooral om het gesprek na het spel over hun eigen product.

Hoe ben je begonnen met het maken van het Technical debt spel?

Het is begonnen met een soort van Monoploy spel waarbij het gaat om snel versus goed. Eerst heb ik iets in Powerpoint in elkaar geknutseld. Dit heb ik gespeeld met mensen, veel feedback op gekregen en deze weer verwerkt in een nieuwe versie. Op een gegeven moment kreeg ik de tip 'Laat het designen' en dat heb ik ook direct gedaan. Jomiro Eming heeft het design speciaal voor dit spel gemaakt. Daarnaast heeft hij het ook nog op zo'n manier gemaakt zodat je alles zelf kan uitprinten in elkaar kan vouwen.
Het spel is beschikbaar op Github waar je het zelf kan downloaden en uitprinten. Het enige wat je zelf moet regelen, is een dobbelsteen.
Ik krijg nog regelmatig feedback en suggesties op variaties van het spel.

Een soort van expansion pack op het spel?

Er staan nu twee versies van de spelregels op Github. Daarom heb ik het op Github gezet. Je kan zelf eventueel een pull request doen. Als is dat wel lastiger met die Adobe design files.

Vaak zijn het de developers die de problemen als eerste zien aankomen. Zij weten het beste hoe stabiel de systemen draaien en wat ervoor nodig is om de snelheid en/of schaalbaarheid van de systemen te verhogen.

Heb je nog tips voor developers om technical debt op de agenda bij het management te krijgen?

Kleine dingen kunnen ze in het team zelf oplossen. Stem dit af met je Product Owner want je zal dan wel moeten uitleggen waaraan je gaat werken.
Grote wijzigingen zal je moeten afstemmen met het C-level management.

Hoe krijg je zo'n grote wijziging verkocht bij het management?

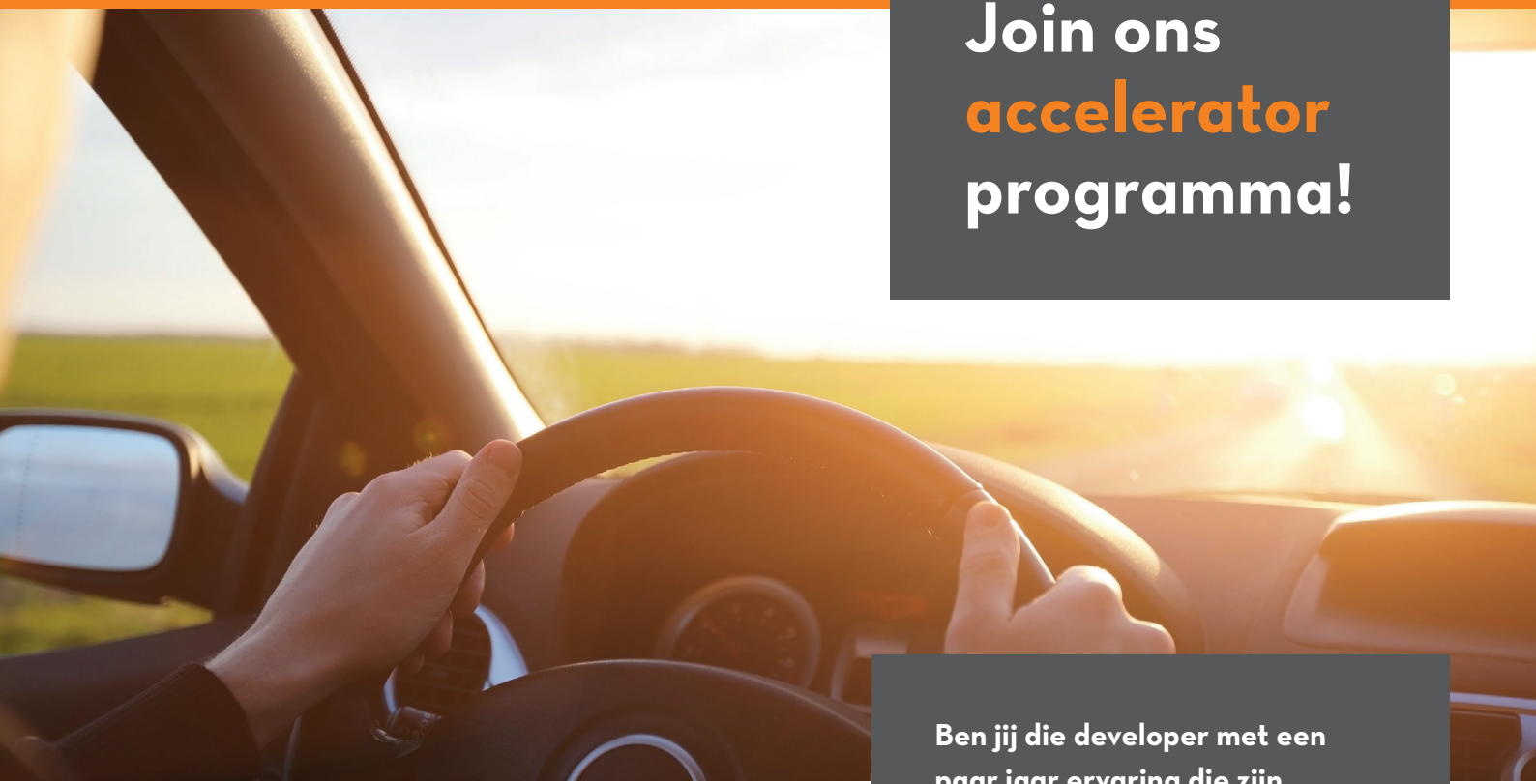
Dit kan je "verkopen" bij hun door aan te geven dat ze anders de lange termijndoelen (van het bedrijf) niet zullen halen. De promised and proven value van een bedrijf kan je uitrekenen. Hierdoor kan je zichtbaar maken wat een bedrijf aan omzet zal mislopen wanneer het systeem niet wordt aangepast om de lange termijndoelen te ondersteunen.
Zelfs als het gaat over grote wijzigingen, kan je beginnen met een Proof of Concept. Je hoeft niet altijd een enabler-team op te zetten om grote wijzigingen door te voeren. Soms kan het ook door een percentage aan capaciteit van de huidige teams te gebruiken om eraan te werken.

Het Technical debt spel is te down*loaden op: https://github.com/arjen-deruiter/techdebtgame*

Voor sponsorships neem contact op met Richelle Bussenius - rbussenius@reshift.nl

“Heb jij graag het stuur in handen?
Met ons op maat gemaakte programma
zet je de volgende stap in je carrière!”

Join ons accelerator programma!



ORDINA

Software Development

- 1 jaar intensieve training voor en door developers
- Uitsluitend moderne technieken
- Combinatie van vakinhoud en soft skills

Ben jij die developer met een paar jaar ervaring die zijn .Net kennis naar het ‘Next Level’ wil brengen?

Word collega bij Ordina Software Development en doe mee aan dit leerzame en leuke traject!

Wil je meer informatie? Kom naar onze stand op Future Tech of mail naar barbara.pronk@ordina.nl

Microsoft • .NET6 • AZURE • DevOps • C# • OpenSource • Cloud • AI/ML • Security • PowerPlatform